# AUDIO ORACLE: A NEW ALGORITHM FOR FAST LEARNING OF AUDIO STRUCTURES

*Shlomo Dubnov*
UCSD, Music Department
La Jolla, CA.
sdubnov@ucsd.edu

*Gérard Assayag*
Ircam - UMR CNRS STMS
Paris, France.
assayag@ircam.fr

*Arshia Cont*
UCSD, Music Dept. and Ircam - UMR CNRS STMS.
cont@ircam.fr

**ABSTRACT**

In this paper we present a new method for indexing of audio data in terms of repeating sub-clips of variable length that we call *audio factors*. The new structure allows fast retrieval and recombination of sub-clips in a manner that assures continuity between splice points. The resulting structure accomplishes effectively a new method for texture synthesis, where the amount of innovation is controlled by one of the synthesis parameters. In the paper we present the new structure and describe the algorithms for efficiently computing the different indexing links. Examples of texture synthesis are provided in the paper.

## 1. INTRODUCTION

The need to find sub-clips in a recording is important for a variety of applications. In case of audio retrieval, question of partial similarity can be used to ask questions whether certain sound clip occurs in a recording, and if it does, where? It can be used for efficient editing, such as jumping to similar sounding places in a recording, selection of audio sub-clips from databases of longer clips, or concatenative synthesis. Reshuffling of sub-clips can be used to create new variants of existing recordings, such as in the case of texture synthesis [1],[2],[3] and audio mosaicing [4]. In this paper we present a new method for indexing of audio data in terms of detection of approximately repeating sub-clips of variable length that we call *audio factors*. The method is motivated by a similar technique used for fast indexing and search in text, called *Factor Oracle*. This method has been used for music generative purposes using MIDI data [5]. The current paper effectively extends the method in several aspects:

- It operates on audio recordings, with notion of similarity being determined by the type of features chosen to represent the audio signal
- It allows approximate matching between feature vectors, effectively controlling a tradeoff between precision of single frame matching and the cumulative similarity / dissimilarity in matching longer sub-clips.

The new structure, called *Audio Oracle (AO)*, accomplishes effectively a new method for concatenative synthesis, where each recombination within clips shares a common *context* between the end of the current segment and its continuation in a new clip. In concatenative systems (e.g. [6] for music and CHATR [7] for speech), units in a synthesis database can be considered as a state transition network in which the state occupancy cost is the distance between a database unit and a target, and the transition cost is an estimate of the quality of concatenation of two consecutive units. This framework has many similarities to HMM-based speech recognition, which needs to have a predetermined architecture and definition of hidden states and their probabilities. Moreover, it has a fixed dependency on the past due to Markov assumption. AO in contrast has no hidden states (which possibly reduces its power) and allows long continuations by following suffix rules. It also should be noted that the meaning of states in concatenative systems (like CHATR) is different from AO, where the states are frames rather then hidden or latent representations. Moreover, the number of states in AO is significantly larger than in concatenative, since every frame is considered a state, effectively grouped *on the fly* according to the distance measure and threshold. The two main applications of concatenative synthesis in computer music applications are audio texture synthesis and audio mosaicing. In the case of audio textures, the common goal of synthesis is to produce more variants of sounds that are perceived as if they were produced by the same sound source, basically extending the amount of sonic material that can be derived from a single recording. In such cases little or no control over the synthesis procedure is required. Existing methods to manipulate natural grains employ basic parametric control over the size or density of the grains, as well as over the amount of randomness or jitter in their recombination [8]. In the case of audio mosaicing, the sound segments are chosen to match a specific goal, such as finding and reordering clips in a manner that correspond best to a given target sound [4]. One of the main challenges in mosaicing is unit selection, i.e. determining the best sub-clips that are both "natural" and that match the target in terms of some set of features. The difficulty resides in the fact that satisfying target constraints severely limits the set of candidate sub-clips that are available in the source. In this paper we describe the use of AO for the first type of concatentative applications, namely texture

synthesis. We show that the strength of the AO algorithm is exploited in terms of its automatically finding the best possible smooth concatenation between different location in the original recording. In a companion paper we introduce matching into AO operation, allowing concatenative synthesis that is *guided* by another sound that acts as a query, operating in a way that is suitable for mosaicing and other data-driven concatenative synthesis applications [9]. In the rest of this paper we present the new AO structure and describe the algorithms for efficiently computing the different indexing links. Examples of texture synthesis are provided in the last section of the paper.

## 2. DESCRIPTION OF THE ALGORITHM

In this section we describe the *Audio Oracle* algorithm. As mentioned earlier, the method is motivated by a similar technique used for fast indexing of symbolic data such as text called *Factor Orcale* [10]. Factor Oracle has been successfully applied to symbolic music data with applications to automatic music improvisation [5] and automatic style imitation on MIDI signals. Audio Orcale is basically an extension of Factor Oracle for continuous data flow such as audio. We first briefly describe Factor Oracle and then move on to the extension for Audio Oracle and discuss the output by showing an example.

### 2.1. Factor Oracle

Basically a factor oracle $P$ is a finite state automaton constructed in linear time and space in an incremental fashion. A sequence of symbols $S = \sigma_1, \sigma_2, ...\sigma_n$ is learned in such an automaton, whose states are $0, 1, 2...n$. There is always a transition arrow (called the factor link) labelled by symbol $\sigma_i$ going from state $i-1$ to state $i, 1 < i < n$. Depending on the structure of $s$, other arrows will be added to the automaton. Transitions directed from state $i$ to state $j$ that belong to the set of factor links and are labelled by symbol $\sigma_k$, are denoted by $\delta(i, \sigma_k) = j$. Some transitions are directed "backwards", going from a state $i$ to a state $j$ and are called suffix links, bearing no label. A Suffix link in oracle $P$ that points from state $m$ to an earlier state $k$ is denoted by $S_P(m) = k$. The factor links model a factor automaton that for every factor $p$ that is a sub-string in $S$ reaches some state by following a unique factor link path labeled by $p$, starting in 0. For every $p$, the reached state corresponds to the end point of that factor appearing in the sequence $S$. Suffix links have also an important property : a suffix link goes from $i$ to $j$ iff the longest repeated suffix of $s[1..i]$ is recognized in $j$. Thus, suffix links connect repeated patterns of $s$. The oracle is learned on-line and incrementally. For each new entering symbol, a new state $i$ is added and an arrow from $i_1$ to $i$ is created with label $\sigma_i$. The algorithm, then, updates the transition structure by iteratively following the previously learned structure backwards through available factor links and suffix links in order to create new ones. Details of Factor Oracle is discussed in [10].

### 2.2. Audio Oracle

Audio Oracle accepts a continuous (audio) stream as input, transforms it into a sequence of feature vectors and submits these vectors to AO analysis. AO outputs an automaton that contains pointers to different locations in the audio data that satisfy certain similarity criteria, as found by the algorithm. The resulting automaton is passed next to the audio generation module, that will be described in the next section. The generative module outputs a new audio stream that is created by concatenation of the audio frames corresponding to AO states that were traversed during the generative process.

To assure modularity, the nature of the data presented to the algorithm is independent of the AO itself and can be any vector per audio frame containing audio features (or combinations of them) that describe the audio segments. Therefore, symbols in Factor Oracle corresponding to each state $i$ of the algorithm are vectors representing a user-defined description of the audio stream. In order to construct the oracle structure, a *distance function* is required that measures the similarity between these "symbols" in the oracle. Defining the distance function is done by the user in a modular fashion as well. For the experiments described in this paper, we define the distance between two vectors $x_1$ and $x_2$ as the Euclidian norm of the difference between the vectors $||x_1 - x_2||$. This distance measure has proven to be fast and sufficient for many applications using Audio Oracle, especially in the case of cepstral vectors as the audio features, as described in the next section. Given this measure, a simple thresholding can determine whether two "symbols" can be considered similar or not in order to construct the state-space underlying the audio structure.

Algorithms 1 and 2 demonstrate psuedo-codes for Audio Oracle construction. During the online construction, the algorithm accepts audio frame descriptions (user-defined audio features) as vectors $\sigma_i$ for each time-frame $i$ and updates audio oracle in an incremental manner. Algorithm 1 shows the main online audio oracle construction algorithm. Algorithm 1 calls the function `Add-Frame`

**Algorithm 1** On-line construction of Audio Oracle

**Require:** Audio stream as $S = \sigma_1\sigma_2\cdots\sigma_N$
1: Create an oracle $P$ with one single state 0
2: $S_P(0) \leftarrow -1$
3: **for** $i = 0$ to N **do**
4: $Oracle(P = p_1\cdots p_i) \leftarrow$ Add-Frame $(Oracle(P = p_1\cdots p_{i-1}), \sigma_i)$
5: **end for**
6: **return** Oracle $(P = p_1\cdots p_N)$

described in algorithm 2 which updates the audio oracle structure using the latest received frame descriptions. This function works very similar to the description of Factor Oracle in section 2.1 except that (1) it accepts *continuous* data flow rather than symbolic data, (2) does not assign symbols to transitions and instead each state has a one-to-one correspondence with frames in audio buffer, and (3)

it uses a distance function (described earlier) along with a threshold $\theta$ to assign the degree of similarity between frame descriptions. The set of links in Audio Oracle are forward arrows $\delta(i, \sigma)$ and suffix links $S_p(k)$.

**Algorithm 2** `Add-Frame` function: Incremental update of Audio Oracle

**Require:** Oracle $P = p_1 \cdots p_m$ and Audio Frame description vector $\sigma$

Create a new state $m + 1$
Create a new transition from $m$ to $m + 1$, $\delta(m, \sigma) = m + 1$
$k \leftarrow S_P(m)$
**while** $k > -1$ **do**
  Calculate distances between $\sigma$ and $S$
  Find indexes of frames in $S$ whose distances from $\sigma$ are less than $\theta$
  **if** There are indexes found **then**
    Create a transition from state $k$ to $m + 1$, $\delta(k, \sigma) = m + 1$
    $k \leftarrow S_P(k)$
  **end if**
**end while**
**if** $k = -1$ (no suffix exists) **then**
  $s \leftarrow 0$
**else**
  $s \leftarrow$ where leads the *best* transition (min. distance) from $k$
**end if**
$S_{p\sigma} \leftarrow s$
**return** Oracle $P = p_1 \cdots p_m \sigma$

Similar to the Factor Oracle algorithm, forward transitions or factor links correspond to states that can produce *similar patterns* by continuing forward and suffix or backward links correspond to states that share the *largest similar sub-clip* in audio buffer.

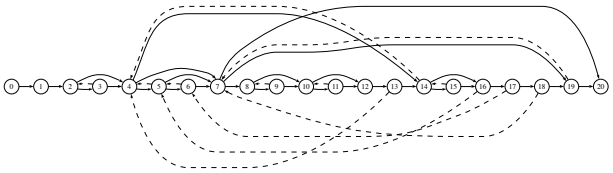
Audio Oracle's complexity is both linear in time and space (independent of the complexity of audio feature calculations). This computational complexity allows the algorithm to work in interactive and real-time environments.

### 2.3. Audio features

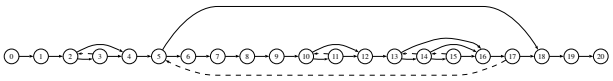
As mentioned earlier, the core Audio Oracle algorithm is independent of the audio feature representation and audio descriptions can be defined by the user. In this section, we focus on one set of audio features that has proven to be useful for audio and music similarity experiments. A common representation of audio results from transforming the time signal into the spectral domain using the Short-Time Fourier Transform (STFT) and taking its absolute or log-absolute values. Another common representation of spectral contents of audio signals is by means of cepstral coefficients. Cepstral analysis provides a method for separating out spectral information from the pitch or individual partials information. Using only the few first coefficients of the cepstrum, the cepstral components related to "spectral envelope" are retained. One of the important properties of the cepstrum is that two signals can be compared by computing Euclidian distance between lower cepstral coefficeints that represent spectral envelope of two sounds. We use Euclidian measure in AO Algorithm 2, as described in the previous section. For the case of general audio, a modification of the cepstrum that uses Mel-Freqeuncy scale is commonly used.

## 3. SAMPLE RESULT

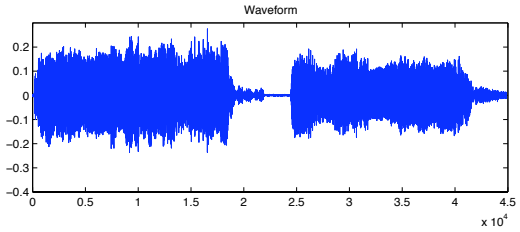
Figure 1 shows a sample run of Audio Oracle algorithm and the state spaces obtained out of the audio using Cepstral Coefficients as features. Figure 1(c) shows the (natural recording) waveform of audio clip used in the experiment which corresponds to a *Blue Jay* bird sound. As seen in the waveform, the bird sound can be heard twice. For demonstration purposes, figures 1(a) and 1(b) show the obtained state space out of Audio Oracle algorithm and for two different threshold $\theta$ values on the distance function. Naturally, a lower threshold means that similarity decisions are much tighter and thus, less forward and backward links would be produced. The numbering on each node correspond to a specific time-frame analysis over the original audio (total of 20 frames of size $93ms$ with $46ms$ overlap). The second occurence of the bird sound appears around frame number 12. Clearly, we would expect suffix links after this state to refer at some point to the earlier occurence since both are somewhat similar. Forward transitions also connect states that share a similar leading frame using the given distance function and threshold.



(a) Learned Audio Oracle with $\theta = 0.04$



(b) Learned Audio Oracle with $\theta = 0.02$




(c) Sample Audio Waveform

**Figure 1**. Audio Oracle Example: (a) and (b) two learned AOs, (c) audio waveform

## 4. APPLICATIONS TO TEXTURE SYNTHESIS

An important property of AO is its power of generation. Navigating the oracle and starting in any place, following forward transitions generates a sequence of indexes in the audio buffer that correspond to repetitions of portions of the audio given the audio feature being used; following one suffix link followed by a forward transition generates an alternative path in the sequence, creating a recombination based on a shared suffix between the current state and the state pointed at by the suffix link. This shared suffix link is called context in context-inference models. In addition to completeness and incremental behavior of this model, the best suffix is known at a minimal cost of just following one pointer. By following more than one suffix link before a forward jump or by reducing the number of successive factor link steps, we make the generation less resemblant to the original. Audio examples demonstrating the synthesis procedure are available on the internet [1] .

### 4.1. AO generative algorithm

Algorithm 3 demonstrate the incremental generative algorithm once an AO is learned. The generation proceeds

**Algorithm 3** `AO-Generate` function:

**Require:** Oracle $P = p_1 \cdots p_m$ in active state $i$, generated audio stream $V = v$ and continuation parameter $0 \leq q \leq 1$

Generate uniformly distributed random number $u$
**if** $u < q$ **then**
  $v \leftarrow vp_{i+1}$
  $i \leftarrow i + 1$
**else**
  Choose at random a symbol $\sigma \in \{\sigma_j \mid \delta(S(i), \sigma_j) \neq nil\}$
  $i \leftarrow \delta(S(i), \sigma)$
  $v \leftarrow v\sigma$
**end if**
**return** Audio stream $V = v$

with probability $q$ to duplicate a sub-clip of the original sound as instructed by AO. With probability $1 - q$ the algorithms jumps back in AO along a suffix link, arriving in a state where a maximal suffix of $v$ that fulfills the minimal distance constraint is recognized. From that state in AO a random choice is made among all possible forward transitions. These transitions indicate which symbols can possibly follow that suffix of $v$. The probability variable $q$ controls the rate of introducing random recombinations in the concatenative procedure, with $q$ close to 1 leading to large sections of $S$ being duplicated in $V$. When $q$ is close to 0, suffix links will be mostly used for generation, resulting in a bigger rate of recombinations with regard to $S$. Corresponding audio buffers can be resynthesized easily by using a simple windowed overlap-add algorithm that assures phase continuity and reconstruction.

[1] `www.cosmal.ucsd.edu/arshia/AO/`

## 5. DISCUSSION

The AO effectively extends the FO to arbitrary sequences of vectors with a given distance function between pairs of vectors. This is done by introducing the following changes in the online FO algorithm: forward arrows are created from suffix link states to a new state if the distance between them is less then a prescribed threshold, and a new suffix link is created from the new state to the best transition in the past. The new structure provides a set of links between all possible recombinations of audio data that are similar enough in terms of some distance measure. It should be noted that our approach is different from other segmental or unit selection methods since there is no hard decision made about "natural" sound units. In generative mode, AO is used to create new sequences of states by following a suffix link and then moving forward to any possible continuation – motivating its application for texture synthesis. Navigating in the state-space provided by the algorithm, provides state sequences where each state is in one-to-one correspondence with an audio frame in a sound file or audio buffer. The *non-linear* sequence generated out of Audio Oracle can be described as a concatenation of segments of sounds. The compact structure of Audio Oracle for a given sound clip, based on sub-clip similarities within the audio file, assures perceptual continuity of the resynthesized audio.

## 6. REFERENCES


[1] G. Strobl, G. Eckel, and D. Rocchesso. Sound texture modeling: A survey. In *SMC*, Marseille., 2006.

[2] L. Lu, L. Wenyin, and H.J. Zhang. Audio textures: Theory and applications. *IEEE Transactions on Speech and Audio Processing*, 12:156–167, March 2004.

[3] S. Dubnov, B. Ziv, E.Y. Ran, D. Lischinski, and M. Werman. Synthesizing sound textures through wavelet tree learning. *IEEE Comput. Graph. Appl.*, 22(4):38–48, 2002.

[4] Ari Lazier and Perry Cook. MOSIEVIUS: Feature driven interactive audio mosaicing. In *DAFx*, London, 2003.

[5] G. Assayag and S. Dubnov. Using factor oracles for machine improvisation. *Soft Computing*, 8-9:604–610, 2004.

[6] D. Schwarz. Corpus-based concatenative synthesis. *IEEE Sig. Proc. Mag.*, 24(1), March 2007.

[7] A.W. Black and P. Taylor. CHATR: a generic speech synthesis system. In *Proceedings of COLING'94*, volume II, pages 983–986, Kyoto, Japan, 1994.

[8] R. Hoskinson and D. Pai. Manipulation and resynthesis with natural grains. In *ICMC*, pages 338–341, San Francisco, 2001.

[9] A. Cont, S. Dubnov, and G. Assayag. Guidage: A fast audio query guided assemblage. In *ICMC*. Copenhagen, September 2007.

[10] D. Allauzen, M. Crochemore, and M. Raffinot. Factor oracle: A new structure for pattern matching. In *Conference on Current Trends in Theory and Practice of Informatics*, pages 295–310, 1999.