Guided Music Synthesis with Variable Markov Oracle

Cheng-i Wang and **Shlomo Dubnov**
Computer Music, Music Department, UCSD
La Jolla, CA 92037, U.S.A
{chw160, sdubnov}@ucsd.edu

Abstract

In this work the problem of guided improvisation is approached and elaborated; then a new method, *Variable Markov Oracle*, for guided music synthesis is proposed as the first step to tackle the guided improvisation problem. *Variable Markov Oracle* is based on previous results from *Audio Oracle*, which is a fast indexing and recombination method of repeating sub-clips in an audio signal. The newly proposed *Variable Markov Oracle* is capable of identifying inherent datapoint clusters in an audio signal while tracking the sequential relations among clusters at the same time. With a target audio signal indexed by *Variable Markov Oracle*, a query-matching algorithm is devised to synthesize new music materials by recombination of the target audio matched to a query audio. This approach makes the query-matching algorithm a solution to the guided music synthesis problem. The query-matching algorithm is efficient and intelligent since it follows the inherent clusters discovered by *Variable Markov Oracle*, creating a query-by-content result which allows numerous applications in concatenative synthesis, machine improvisation and interactive music system. Examples of using *Variable Markov Oracle* to synthesize new musical materials based on given music signals in the style of Jazz are shown.

1 Introduction

Machine Improvisation systems have become powerful tools for artificially augmented performance where machines provide some of the creativity to the musical outcome. With the use of machine learning and algorithmic techniques, the need for programming the compositional algorithms governing the artificial musical partner has become less burdening, and the need for programming is largely substituted by providing musical examples from which the machine extracts patterns or rules. Furthermore, it is now possible to rehearse or perform with machines that can capture the style of the live musician either from the musical contents or based on pre-recorded examples to create improvisations.

As in any musical ensemble, the contribution of a partner lies not only in his creativity and virtuosity as a soloist, but also in his ability to listen to the companions. Moreover, to help such a coordination, it is possible for a group of performers to agree ahead of time on some musical structures, that could be more or less tight and depending on the creative choices of the participants. In our pervious experiments with Machine Improvisation, the computer was used to capture and create stylistic re-injections into a duo human-machine performance, where the computer was able to extract significant musical structure from the live musician's input and creates variations that augment the soloist in various unexpected, yet stylistically coherent ways (Dubnov and Assayag 2013; Assayag et al. 2006). The creative choices of the machine were controlled by a human operator who could specify regions or select the dimensions of music materials that the machine improvises on, as well as has some high level control on the rate of recombinations or variations. Recently, to alleviate the need for a human operator and to endow the machine with greater autonomy, we have introduced functions that would allow tighter integration between the machine musical output and the human musicians. In some sense, what we want to accomplish is to make the machine listen to its human partners and respond in a musically meaningful way by guiding its generative mechanism towards materials that conform to both the content and style of the live performance. To achieve such a goal, we propose a new guided synthesis algorithm using *Variable Markov Oracle*, which is a variant of *Audio Oracle* (Dubnov et al. 2007; Dubnov, Assayag, and Cont 2011) method that had been already used and tested extensively in improvisation situations. The current paper extends earlier experiments on querying the target oracle with a single frame to create "hot spots", i.e. marking desired locations along the possible trajectories for the algorithm to arrive at during improvisation (Surges and Dubnov 2013), to use a sequence of frames as query for the target oracle to match to. As the first step, the experiments presented in the paper are done mostly offline. In this case, the system could be considered more as a meta-compositional tool, but with some extra technical challenges, the system has the capability to be ported to a real time setting for use in live improvisations. Such future goals will be discussed later in the paper.

2 Background and Related Works

The problem of planning and control that includes improvisation capabilities has also become an area of interest in

other domains. Specifying control signals to guide a system into desired behavior is common in robotics and other dynamic systems, where adding a randomized strategy from examples might add more flexibility to a system (Donze et al. 2013). In the case of music, the need of the improvisation to conform to outside constraints, such as partnering with other musicians, is less strictly defined and is with no critical safety specifications. However, undesired notes can still be quite annoying, especially during live performances where machines are involved. In music alignment problem, solutions are proposed to match the same music piece in different media such as audio, midi, score, etc (Ewert, Müller, and Dannenberg 2011), but the solutions were focusing on alignment, not creation. Another approach to this problem is introducing constraints to the improvisation system, as done in (Pachet and Roy 2011; Roy and Pachet 2013). Other related examples that use time-automata to provide flexible and even improvisatory performances in coordination with a plan are the interactive score projects, such as Antescofo (Echeveste et al. 2013) and Virage (Allombert et al. 2010). In such cases, a formal score specification allows the system to modify its performance according to expressive inflection or a musician, mostly limited to time changes within a tightly predetermined sequence of events, thus making the system difficult to navigate the specification in a highly non-linear temporal fashion, i.e. allowing jumps and recombination of events in a way that the proposed *Variable Markov Oracle* (*VMO* hereafter) is designed for.

Taming of the randomly generative oracle is both a creative and a scientific task. In some respect, the operation of the *Audio Oracle* (*AO* hereafter) and similar systems that capture musical structure and generalize the creative process through random permutation of permissible musical structures, presents a phase of Blind Variation (BV) in a creative process (Campbell 1960). To achieve a truly creative outcome, such a process has to also include a phase of Selective Retention (SR), or making choices as to which new possibilities that the BV phase uncovered are actually going to serve the more tightly specified and directed goal of the overall artistic outcome. It should be noted that in order for the SR phase to succeed, the divergent BV phase has to be already relatively structured. Or, in other words, it is blind but not random; i.e. it has the ability to create meaningful alternatives which are not guided towards a specific goal. The problem of guided oracle navigation is addressing this second phase, after the basic structure of the musical style "hidden" in the musical recording has been already uncovered by the *AO* process. To allow for such control, we had to modify the structure of the *AO* to expose some of the hidden states or underlying similarities of musical materials for an outside query. Accordingly we name the newly proposed method by *VMO*, since the probabilities of navigating the oracle structure, as will be explained in the paper, will now be modified by the query. *VMO* is somewhat conceptually similar to Markov Decision Processes approaches, but without a training phase and rather focusing on immediate navigation. The term "*Variable*" is used since the oracle structure captures variable length context dependency by tracking repeated sub-clips of the input sequence.

VMO is along the line of research in *Factor Oracle* (*FO* hereafter)(Allauzen, Crochemore, and Raffinot 1999) and *AO*. The current research of *VMO* extends previous research result in *AO* from having single frame query function to having audio signal sequence query-matching functionality by explicitly exploring how feature frames in an audio signal are clustered together by the oracle structure. The construction algorithm for oracle structures allows real-time construction and enables fast retrieval and recombination of sub sequences (factors) of the original audio signal. In this work, the newly devised construction and query-matching algorithms of *VMO* are introduced. The paper is organized as followed, in section 3, the problem and significance of guided improvisation problem are described and elaborated; in section 4, firstly a brief background on *FO* and *AO* is provided. Then the main method for constructing *VMO* is presented. Lastly the query-matching algorithm utilizing *VMO* is elaborated. In section 5, two examples of using *VMO* and the query-matching algorithm to create new jazz music materials are shown. One example is guided synthesis in which the query is a lead saxophone recording to guide the synthesis of an accompaniment. The other example is to use the accompaniment as a query to guide the synthesis of a lead saxophone. Conclusions and future works are provided in section 6.

3 The Guided Improvisation Problem

In our experience with machine improvisation, we have encountered several scenarios where interactions between the machine and the human input are required; i.e. the machine improvises along with another human musician, and we refer to such machine improvisation as guided improvisation. We found that specifying the design requirement of guided improvisation is best done using musical terms. The simplest case of improvisation control and interaction appear in what we term as the "cadence" problem , which is how to make the machine improvisation reach an ending point together with the human musician automatically. We have already used a simple single chroma query to lead *AO* to a certain tonal area where it matches the soloist so we could stop *AO* together with the musician in a situation where both are chromatically consistent with respect to each other (Surges and Dubnov 2013). In these applications, the musician would use a single long note as a query to a system that would emphasize regions in the oracle that have a related chroma.

Another generalized version of the "cadence" situation is the "queueing" problem, i.e. how to trigger changes in the selection of materials that the oracle structure is improvising on in ways corresponding to the musical input. In the previous experiments, we could not distinguish among types of musical materials simply based on a single chroma or note. Moreover, changing the note or replacing the query in time creates a "moving target" situation, that could indeed fall into a desired trajectory, but without any assurances or algorithmically efficient solutions. Therefore, we wanted to allow the system to listen to more than one single note, and to be able to switch between regions or alternative oracles

based on a longer query. In this case, the sequential nature within the query is taken into account. Intuitively, to make regions distinct we need a "longer" index then a single note query, especially if we would like to consider adding transformations, such as transposition or pitch shift in the future. So identification of melodic phrases or even timbral "gestures" can be used to switch regions in more general ways than the cadence problem case.

Operation of the guided improvisation can be also understood in terms of ensemble and larger compositional relation between human and artificial partners. In an "accompaniment" scenario, the oracle is trained on a selection of musical materials that support a live solo or lead musical instrument. One can think of it as a "music -1" situation, where a solo input is driving an intelligent and creative playback of an accompaniment piano or even a complete Jazz ensemble. The role of the system then would be instantaneously harmonizing, providing a bass line and even rhythm to a melodic or polyphonic lead query. The example presented in the paper for this type of guided improvisation is example I, presented in section 5, where the accompaniment mechanism is simplified to recombination of recorded full-band jazz audio frames.

A different configuration and musical use of the proposed system is for constraining a machine generated solo for a given song or standard. In such case, a recording of the backing ensemble is provided as a query into an oracle trained on an unrelated solo. The purpose of the query is to navigate a different solo in a way that the solo licks would be extracted in a fashion that matches the song composition. In such case the query is used to create the solo, while the harmonic and rhythmic grids are fixed in the query track. The example for this configuration of guided synthesis is example II in section 5. Taking turns or switching roles between lead and support or between query and oracle is of course possible throughout the course of the composition, and could be specified by an external score or script that plans the interactions ahead of time.

In short, as a next step to address the aforementioned "cadence" and "queueing" problem from our previous result in *AO*, a revised oracle structure, *VMO*, is introduced to allow querying with a sequence instead of a single frame. Although the examples presented in this work are created in an off-line fashion which makes the guided improvisation problem into a guided synthesis one, we consider the current work to be the necessary step toward real-time application and implementation.

4 Algorithms

In this section the description of the *VMO* algorithm is provided. As mentioned in section 1, *VMO* is the new branch grown out of *FO* and *AO*. Before getting into the details of *VMO*, a compact introduction to *FO* and *AO* is provided in this section.

FO is a variant of suffix tree that aims at fast indexing and retrieval of repeated sub-strings (factors) of a symbolic sequence. *FO* could also be viewed as a finite state automaton constructed in linear time and space with an incremental approach. For a sequence of symbol $Q = q_1, q_2, \ldots, q_t, \ldots, q_T$, an *FO* is constructed with T states and each symbol q_t is associated with a state. Two kinds of links, forward link and suffix link, are created during the construction of *FO*. Two types of forward links are presented in the structure; the first is an internal forward link which is a pointer from state $t-1$ to t labeled by the symbol q_t, denoted by $\delta(t-1, q_t) = t$. The other forward link is an external forward link which is a pointer from state t to $t+k$ labeled by q_{t+k} with $k > 1$. An external forward link $\delta(t, q_{t+k}) = t+k$ is created when

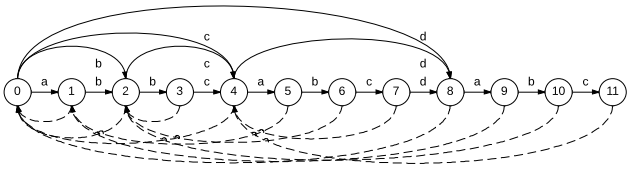


Figure 1: *FO* of Q = "abbcabcdabc". Dashed arrows are the suffix links and normal arrows are the forward links with labels of each symbol.

$$\begin{aligned} q_{t+1} &\neq q_{t+k} \\ q_t &= q_{t+k-1} \\ \delta(t, q_{t+k}) &= \emptyset. \end{aligned}$$

In other words, an external forward link is created when the newly added internal forward link is unseen for previous occurrence of q_t. The function of the forward links is to provide an efficient way to retrieve any of the factors of Q, starting from the beginning of Q and following a unique path formed by forward links.

Suffix link is a backward pointer that links state t to $k, t > k$, without a label and is denoted by $sfx[t] = k$.

$$sfx[t] = k \iff \text{the longest repeated suffix of } \{q_1, q_2, \ldots, q_t\} \text{ is recognized in } k.$$

Suffix links recognize repeated patterns in Q. The construction of *FO* could be done incrementally as new symbol appearing and appending to the end of Q. The algorithms for constructing *FO* are provided in (Lefebvre, Lecroq, and Alexandre 2003). An example *FO* structure is depicted in figure 1. The example will be further exploited in section 4.3 to explain the decoding steps in the query-matching algorithm proposed in section 4.3.

AO is the continuous extension of *FO*. The input $O[t]$ is a continuous time series sampled at discrete time and $O[t]$ could be multidimensional. To extend the domain of *FO* from symbolic sequences to continuous time series, such as an audio signal, a threshold θ is introduced as a criterion for determining if $O[t]$ is similar to states found in $O[1 \ldots t-1]$ by following suffix links. θ is associated with the metric over the feature space given the signal. Two instances $O[i]$ and $O[j]$ are considered similar if $|O[i] - O[j]| \leq \theta$. The metric should be chosen according to the application area and features used. For the rest of the paper, L_2-norm is used as the metric between feature frames. The details of how *AO* is constructed is provided in (Dubnov et al. 2007) and the

unsupervised way to find θ is shown in (Dubnov, Assayag, and Cont 2011).

4.1 Construction of Variable Markov Oracle

The main contribution of introducing *VMO* is to explicitly identify the clusters of frames formed during the *AO* construction. The clusters are formed by tracking suffix links along the states in an oracle structure. The clusters formed by gathering states connected by suffix links have the following properties; 1) states connected by suffix links are guaranteed to have distances less than θ, 2) clusters related to each other sequentially due to the fact that the cluster to which a state belongs is dependent on its previous state and thus the cluster to which the previous state belongs, 3) each state belongs to only one cluster since each state has only one suffix link.

To explicitly keep track of the clusters and also to maintain the on-line nature of the algorithm, the construction of *VMO* combines *FO* and *AO* in the sense that the sequence of cluster labels are treated as Q, the symbolic sequence, in *FO* construction. And pointers to O are tracked by introducing a list of pointers, $B = [b_1 \dots b_n \dots b_N]$, with N the number of clusters formed and b_n a list containing the pointers (states number, frame numbers) for the nth cluster. In a nutshell, *VMO* accepts O as input and returns an oracle structure keeping track of the cluster label sequence Q and also the lists of pointers to O. The lists of pointers are stored in B and indexed by Q.

Let O be the incoming new signal and t the time index. We use $O[t] = O_t$ to represent the newly observed value or vector at t. Forward link from state i to state j labelled by q is denoted by $\delta(i, q) = j$. Suffix link from state j to state i is denoted by $sfx[j] = i$ without labeling. We use $Q = [q_1, \dots, q_T]$ to denote the label sequence for clusters of observations $O = [O_1 \dots O_T]$. The initialization of *VMO* is provided in Algorithm 1. In Algorithm 2, the incremental algorithm for an incoming signal is provided. For each new incoming samples O_t, a new state is constructed with the internal forward link $\delta(t-1, q_t) = t$ created. The cluster label q_t for O_t is initialized as null. The while loop from line 5 to line 15 in Algorithm 2 is the standard process to assign external forward links and suffix links introduced in (Lefebvre, Lecroq, and Alexandre 2003; Dubnov et al. 2007). Line 16 to line 25 in Algorithm 2 is the newly introduced part of *VMO* that assigns the cluster label to q_t then appends the pointer of O_t to b_{q_t}. In this paper, for the algorithms described in pseudo codes, $X[i]$ means getting the item from an array X in its ith location; $[a; b]$ means appending b to the end of a; $X_{i,j}$ means accessing the ith row and jth column of a matrix X; and $X(i, :)$ means retrieving the whole ith row in a matrix X.

4.2 Determining Threshold via IR

The threshold θ has to be specified before the construction of a *VMO*. In (Dubnov, Assayag, and Cont 2011), it is shown that the θ for constructing an *AO* could be determined by calculating *Information Rate(IR)* over possible θ values, then selecting θ with the highest *IR* value. Since *VMO* inherits all the properties from *AO*, the same approach

Algorithm 1 On-line construction of *VMO*

Require: Time series as $O = O_1 O_2 \dots O_T$
1: Create an oracle P with initial state p_0
2: $sfx_P[0] \leftarrow -1$, $B \leftarrow \emptyset$, $N \leftarrow 1$
3: **for** $t = 1 : T$ **do**
4: $Oracle(P = p_1 \dots p_t) \leftarrow$ Add-Frame$(Oracle(P = p_1 \dots p_{t-1}), O_t)$
5: **end for**
6: **return** $Oracle(P = p_1 \dots p_T)$

Algorithm 2 Add-Frame

Require: Oracle $P = p_1 \dots p_t$, time series instance O_{t+1}
1: Create a new state $t + 1$
2: $q_{t+1} \leftarrow 0$, $sfx_P[t + 1] \leftarrow 0$
3: Create a new transition from t to $t+1$, $\delta(t, q_{t+1}) = t+1$
4: $k \leftarrow sfx_P[t]$
5: **while** $k > -1$ **do**
6: $D \leftarrow$ distances between O_{t+1} and $O[\delta(k, :)]$
7: **if** all distances in D is greater than θ **then**
8: $\delta(k, q_{t+1}) \leftarrow t + 1$
9: $k \leftarrow sfx_P[k]$
10: **else**
11: Find the forward link from k that minimizes D
$k' \leftarrow \delta(k, :)[\text{argmin}(D)]$
12: $sfx_P[t + 1] \leftarrow k'$
13: **break**
14: **end if**
15: **end while**
16: **if** $k = -1$ **then**
17: $sfx_P[t + 1] = 0$
18: Initialize a new cluster with current frame index
$b_{N+1} \leftarrow t + 1$
19: $B \leftarrow [B; b_{N+1}]$
20: Assign a label to the new cluster, $q_{t+1} \leftarrow N + 1$
21: Update number of clusters, $N \leftarrow N + 1$
22: **else**
23: Assign cluster label based on assigned suffix link
$q_{t+1} \leftarrow q_{k'}$
24: $b_{q_{k'}} \leftarrow [b_{q_{k'}}; t + 1]$
25: **end if**
26: **return** Oracle $P = p_1 \dots p_{t+1}$

is applied here. In brief, given the definition of *IR* and let $x_1^N = \{x_1, x_2, \dots, x_N\}$; $H(x)$ stands for the entropy of x,

$$IR(x_1^{n-1}, x_n) = H(x_n) - H(x_n|x_1^{n-1}),$$

the value of *IR* could be approximated by replacing the entropy term with complexity measure associated with a compression algorithm. The complexity measure usually is in bits used to compress x_n and $(x_n|x_1^{n-1})$. In (Lefebvre and Lecroq 2002), a compression algorithm, *Compror*, proven to have similar performance to *gzip* and *bzip2* based on *FO* is provided and the detail formulation of how *Compror*, *AO* and *IR* are combined is provided in (Dubnov, Assayag, and Cont 2011).

4.3 Query-Matching with Markov Oracle

Let R be the query observation indexed by t, and denote $R[t]$ by R_t. The matching algorithm provided in Algorithm 3 takes R as input and matches it to the target *VMO*, P, constructed by a target time series, O. The algorithm returns a recombination path and a corresponding cost. The recombination path corresponds to the sequence of indices that will reconstruct a new sequence from O that best resembles the query. The cost is the reconstruction error between the query and the best match from O given a metric on a frame-by-frame basis.

The query-matching algorithm proposed in this section is a dynamic programming algorithm. The algorithm could be separated into two steps, initialization and decoding. In Algorithm 3, the initialization is in line 1 to line 7. During initialization, the number of clusters N is obtained from the cardinality of B. Then for the nth cluster, the frame within the nth cluster that is closest to the first query frame is found and stored. After the initialization step, the decoding step (line 8∼17 in Algorithm 3) iterates over the rest of the query frames from 2 to T to find N paths, with each path beginning with the state found corresponding to the respective cluster in the initialization step. To find the path starting at the nth cluster at query frame number t, let η be the list storing possible forward transitions in terms of the cluster label indicated by $\delta(t-1,:)$; then the possible frames, b', for t could be retrieved by indexing B with η. The path and cost at time t for the nth path is then determined by the minimum between R_t and $O[b']$. A special point has to be made here about line 10 in Algorithm 3: in line 10 the inclusion of $M_{n,t-1}$ allows the query-matching algorithm to have time-stretching capability when comparing two time series since self-transition is allowed. At the end of the decoding step, the path with the corresponding minimum cost and the cost itself are returned. It could be observed that the proposed query-matching algorithm is similar to Viterbi decoding algorithm for Hidden Markov Model and max-sum inference algorithm for graphical model (Wainwright and Jordan 2008) in the sense that each update in the decoding step depends only on its neighboring findings thus making it efficient to compute and of no need to search over the whole state space. A visualization of Algorithm 3 from initialization to decoding for one path among the N paths is shown in figure 2.

Algorithm 3 Query-Matching

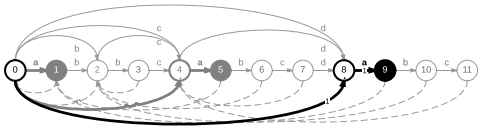
Require: Target signal in *VMO*, P, includes Q the cluster label sequence and O the signal, and query time series $R = [R_1 \dots R_T]$

1: Get the number of clusters, $N \leftarrow |B|$
2: Initialize cost vector $C \in \mathbb{R}^N$ and path matrix $M \in \mathbb{R}^{N\times T}$ with all zeros.
3: **for** $n = 1 : N$ **do**
4: $\quad D \leftarrow$ all distances between R_1 and $O[b_n]$
5: $\quad M_{n,1} = b_n[\mathrm{argmin}(D)]$
6: $\quad C_n = \min(D)$
7: **end for**
8: **for** $t = 2 : T$ **do**
9: $\quad$ **for** $n = 1 : N$ **do**
10: $\quad\quad$ Gather the cluster labels for forward links from $M_{n,t-1}$ and $M_{n,t-1}$ itself. $\eta \leftarrow Q[\delta(M_{n,t-1},:); M_{n,t-1}]$
11: $\quad\quad$ Gather all states from possible clusters, $b' \leftarrow B[\eta]$
12: $\quad\quad D \leftarrow$ all distances between R_t and $O[b']$
13: $\quad\quad M_{n,t} = b'[\mathrm{argmin}(D)]$
14: $\quad\quad C_n \mathrel{+}= \min(D)$
15: $\quad$ **end for**
16: **end for**
17: **return** $M[\mathrm{argmin}(C)], \min(C)$

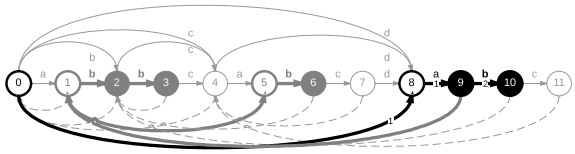
5 Guided Synthesis with Markov Oracle

In this section, Algorithm 3 is used to synthesize new music signals by using a query music signal guiding a path traversing the target music signal. This path matches the target music signal and the query one in the feature space. The target music signal in the guided synthesis application is treated as the material for concatenative synthesis, and the query music signal provides the map to how the materials should be recombined. For the experiments described hereafter, the steps are as followed: first, both the target and query music signals are converted from time-domain waveform to appropriate representations. For this work, Chroma is chosen to be the feature used for all the experiments, and is calculated using a window length of N with $\frac{N}{2}$ overlap, frequency analysis ranging between $f_{min} = 63.5\tilde{4}Hz$ to $f_{max} = 22050Hz$ and 12 bins per octave. Then, the target music signal in Chroma is indexed by *VMO* as O, and the query music signal in Chroma is used as query input, R, for Algorithm 3 to retrieve the recombination path $M[\mathrm{argmin}(C)]$. At last, the recombination path $M[\mathrm{argmin}(C)]$ is used to index the target music signal synthesizing a new music signal with overlap-add method using a window length of N, $\frac{N}{2}$ overlap and hamming window. For the examples shown below, N is set empirically to $2^{15} = 32768$ from listening judgements by the authors. The music signals for the examples are obtained from free shared music recordings in the style of Jazz. Two lead tenor saxophone recordings are cut into segments of $50 \sim 100$ seconds long and are used as query music signals, R. Full band recordings consist of drum, bass, electronic guitar and piano of length $20 \sim 40$ seconds are used as O to construct *VMO*s. Although in theory, the query-matching algorithm could be applied to all genres and features, the combination of Jazz and Chroma is chosen since by far the query-matching algorithm still works on frame level matching between the query and the target music signal, and Jazz music allows more musical meanings to be retained even if the musical structure is broken on a larger scale (such as chord progression or sectional changes) due to the recombination of audio frames from the original audio.

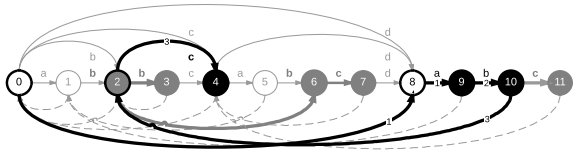
For the guided synthesis application, the self-transition mechanism in line 10 of Algorithm 3 is skipped. Self-transition capability is essential in retrieval tasks, but in the synthesis application it gives undesired results since self-transition allows too much freedom for the matching algo-



(a) At $t = 1$ (Initialization for cluster **a**); $\{\mathbf{a}, 9\}$, the pair of cluster initialized and frame matched. At initialization, for cluster labeled as **a** the choices for first frame are the stored in the list, $b' = \{1, 5, 9\}$, corresponding to b' in Algorithm .3. Assuming the closest frame in O to R_1 with label **a** is O_9, then the first frame for path beginning with cluster **a** will be 9. With the help of keeping track of B, the calculation between R_1 and $\{O_1, O_5, O_9\}$ is straight forward.



(b) At $t = 2$ (Decoding); $\{\mathbf{b}, 10\}$, the pair of cluster identified and frame matched. At $t = 2$, the only possible cluster following cluster **a** from $t = 1$ is **b**, thus making frames in $b' = \{2, 3, 6, 10\}$ the possible candidates. Let O_{10} be the closest frame from $O[b']$ to R_2



(c) At $t = 3$ (Decoding): $\{\mathbf{c}, 4\}$, the pair of cluster identified and frame matched. At $t = 3$, the possible clusters following cluster **b** from $t = 2$ is **b** and **c** by examining the forward links from state 10. The possible frames are now the union of cluster **b** and **c**, $b' = \{2, 3, 4, 6, 7, 10, 11\}$. Let the closest frame from $O[b']$ to R_3 be O_4, the result path beginning at cluster **a** is $\{9, 10, 4\}$. The steps from (a) to (c) are done for all other 3 possible paths as well

Figure 2: Decoding steps: Consider the target time series represented as the *VMO* shown above, the same from figure 1. The light gray parts of each subplot are the same from figure 1. In each subplot, parts marked by black with thick arrows indicate the path for the chosen state, dark gray ones with thick arrows represent possible paths and filled circle represents the candidate states. Numbers on the thick black arrows are step numbers. In this example, the query R, is assumed to have 3 frames and the subplots demonstrate hypothetic steps for the path started with frames in O in cluster labeled by **a** (among 4 possible paths started via **a**, **b**, **c** or **d**). Here the visualization of the query time series is omitted and the path is chose generically to demonstrate Algorithm 3. For brevity and clarity the self-transition mechanism introduced in line 10 of Algorithm 3 is omitted in this figure, but the operation could be done straightforwardly by considering $Q[M_{n,t-1}]$ as candidate cluster to transition to.

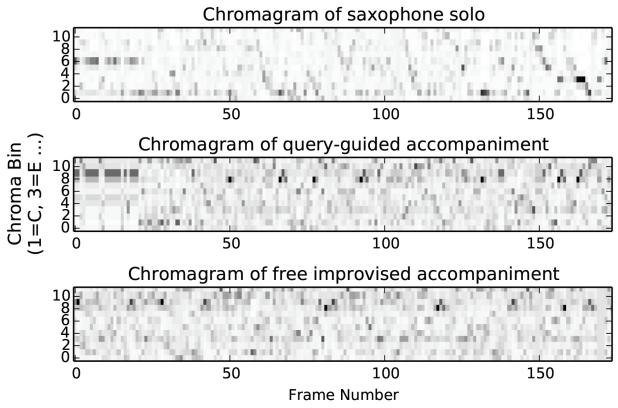


Figure 3: Example I. (Upper) Chromagram of the saxophone lead. (Middle) Chromagram of the query-guided accompaniment. (Bottom) Chromagram of the free improvised accompaniment.

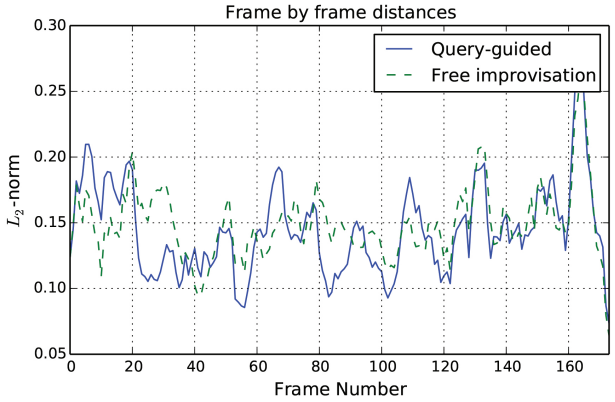


Figure 4: Frame-by-frame L_2-norm between Chromagrams of lead and accompaniment for Example I. The $\{\mu, \sigma\}$ of L_2-norm error for guided-synthesis and free improvisation are $\{0.13, 0.07\}$ and $\{0.15, 0.06\}$ respectively.

rithm to "stutter" in the same state without progressing. To discard self-transition functionality in Algorithm 3, line 10 of Algorithm 3 is replaced with $\eta \leftarrow Q[\delta(M_{n,t-1,:})]$.

To qualitatively evaluate the query-guided synthesis, a free (unguided) improvised accompaniment using the *AO* mentioned in (Surges and Dubnov 2013) is synthesized as well to compare with the query-guided version. Example I is shown here and depicted in Figure 3. To aid the evaluation with quantitative measures, frame-by-frame distances between the Chromagrams of query music signal and the two synthesized accompaniments are plotted in Figure 4. From Figure 4, it is observed that the errors could potentially be correlated. After examination of the parameters used, we conclude that the possible correlation is due to the fact that the query signal used in this case might be too far from the target signal in the feature space, thus leading to rather indistinguishable synthesis between the query-guided

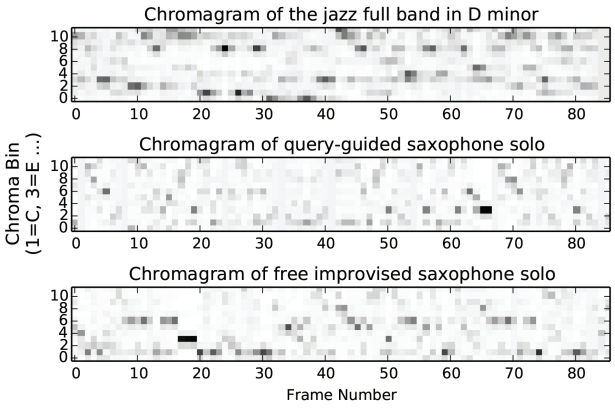


Figure 5: Example II. (Upper) Chromagram of the saxophone accompaniment. (Middle) Chromagram of the query-guided lead. (Bottom) Chromagram of the free improvised lead.

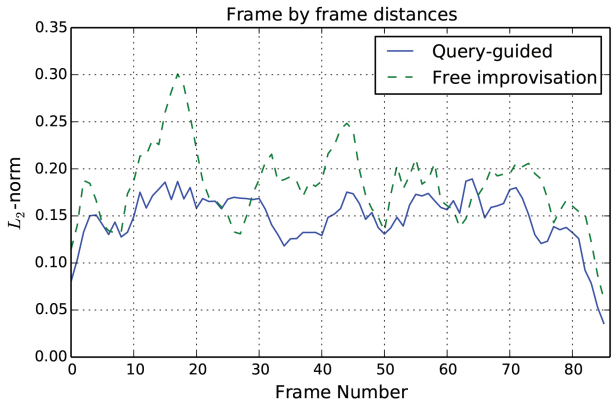


Figure 6: Frame-by-frame L_2-norm between Chromagrams of lead and accompaniment for Example II. The $\{\mu, \sigma\}$ of L_2-norm error for guided-synthesis and free improvisation are $\{0.14, 0.05\}$ and $\{0.18, 0.06\}$ respectively.

and free improvised version from the signal point of view. But in Figure 3, it could also be observed that the query-guided accompaniment did have a more similar structure to the lead than the free improvised one. The phenomenon is clearer at the beginning of the signal. Aurally, when listening to the mix of the lead and the synthesized accompaniments, we observe that the free improvised accompaniment has a better sense of continuation in terms of how sub-clips of the original signals are recombined. In contrast, the query-guided accompaniment sounds more "broken" due to the use of shorter sub-clips to match the query. Nevertheless, in the case of the free improvised accompaniment, the original lead and the accompaniment sound separately from each other even if the tonality of the lead and the original accompaniment recording were chosen to be compatible with each other. In listening to the mix with query-guided accompaniments, the synchronization between the lead and the accompaniment in terms of harmonicity is more obvious. The sound examples could be found at the project page (http://chengiwang.com/projects/MO.php). For the examples, the left channel is always the query, which stays unchanged, and the right channel is the new generated audio either by query-guided (*VMO*) or free-improvisation (*AO*) approach.

After the experiment of using a lead saxophone as the query and the accompaniment recording in *VMO* as target, another experiment, where the roles of the lead and the accompaniment are switched, is performed. Example II is shown here in Figure 5. The L_2-norm errors for example II are depicted in Figure 6. The sound examples for example II are also accessible by the project webpage mentioned above. For example II, similar conclusions to example I based on corresponding plots could be drawn. But with example II, the L_2-norm error plot in Figure 6 reflects the fact that the query-matching algorithm is able to find the path that minimizes frame-by-frame distances between the query and the target. By listening to example II, we find it clearer that the synthesized lead guided by the accompaniment conforms to the harmonic changes and dominant melodic lines of the accompaniment. In general, although aesthetically there is no clear and obvious evidence that using query-guided approach is superior than our previous approach of using unguided machine improvisation, it is still asserted that the materials generated using query-guided approach sound musically meaningful and perform a better job conforming to the query used than the ones generated with free improvisation approach. Extra examples of both cases are accessible via the project webpage.

6 Conclusions and Future Works

In conclusion, the problem of control / guided improvisation is described, and a new method, *VMO*, focused on extending previous results using single frame query to sequence-of-frame query is proposed. The *VMO* method provides the necessary structure for efficient algorithms to be devised for the query-matching task. Audio examples are generated to show the capability of the query-matching algorithm in the context of guided improvisation such that unrelated saxophone leads and jazz accompaniments are used as query and target oracle interchangeably. The results from current experiment show that the newly proposed *VMO* method does provide a path to solve the "queueing problem" mentioned in section 3. The next step for this work is to incorporate mechanisms, such as user defined threshold or criteria, for the query-matching algorithm to "turn off" in the middle of a signal if the query is too far from the target in the feature space and makes the query-matching meaningless. Real-time implementation will also be an important next step which enables practical performance / rehearsal experiments with musicians and composers.

We list some aspects of musical interaction that are not addressed by this work in the following: 1) tight synchronization issues, such as beat matching. Our system is not aware of rhythm, meter or tempo at this point. Accordingly,

although the query might create timing shifts that are natural and even interesting in some genre, it will not work well with music that uses rhythmic patterns as the main expressive tool. 2) In musical practice, synchronized entries of a beat, or tightly triggered tutti sections, are a common artistic tool. Allowing the oracle and the improviser to hit notes together after silence, or trigger notes sequentially in a precisely timed way is a task for future research. 3) Use of harmonic progression / variations / substitutions as a constraint for the oracle can be accomplished indirectly in our current system if the harmonic grid or the chord sequence is used as a query. We consider in the future the possibility of specifying a chord progression symbolically and providing it as a query to a recording. One problem with using such an approach is that harmonic theory and chroma analysis of audio are difficult to reconcile, and even more significantly, should not be directly matched since natural harmonic and melodic processes allowed in the improvisation fall under same chord or harmonic label, especially if we consider chord prolongations, substitution and many other creative techniques that are not explicitly notated in the symbolic representation of Jazz standard.

In general, these issues relate to even a more general problem that we call a "duet problem" that appears when more or less free improvisation happens between two musicians or between a musician and a machine. A related problem in offline composition using Midi / symbolic sequences was encountered earlier in composing "Composer Duets", briefly described in (Dubnov and Assayag 2013). The idea there was to let the oracle try to match a polyphonic pattern from a duet recording. We do not exactly have such situation in audio, since normally we do not have a multi-track audio available for oracle training, so that one track could be used for query while another one for improvisation. We might in the future experiment with creating recordings where two tracks are produced specifically for the accompaniment task. In such case, guiding of the oracle output on one track will be done by querying a second related track.

7 Acknowledgments

Thanks to the support from Center for Research in Entertainment and Learning (CREL) at California Institute for Telecommunications and Information Technology (Calit2) for this research, and thanks to Mr. Greg Surges for the advices on implementation and technical issues.

References

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.

Allombert, A.; Marczak, R.; Desainte-Catherine, M.; Baltazar, P.; Garnier, L.; and Yeti, B. 2010. Virage: Designing an interactive intermedia sequencer from users requirements and theoretical background. In *Proceedings of the International Computer Music Conference*.

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

Campbell, D. T. 1960. Blind variation and selective retentions in creative thought as in other knowledge processes. *Psychological review* 67(6):380.

Donze, A.; Libkind, S.; Seshia, S. A.; and Wessel, D. 2013. Control improvisation with application to music. Technical report, DTIC Document.

Dubnov, S., and Assayag, G. 2013. Memex and composer duets: computer-aided composition using style mixing. *Open Music Composers Book* 2.

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.; Assayag, G.; Cont, A.; et al. 2007. Audio oracle: A new algorithm for fast learning of audio structures. In *Proceedings of International Computer Music Conference (ICMC)*.

Echeveste, J.; Cont, A.; Giavitto, J.-L.; and Jacquemard, F. 2013. Operational semantics of a domain specific language for real time musician–computer interaction. *Discrete Event Dynamic Systems* 23(4):343–383.

Ewert, S.; Müller, M.; and Dannenberg, R. B. 2011. Towards reliable partial music alignments using multiple synchronization strategies. In *Adaptive Multimedia Retrieval. Understanding Media and Adapting to the User*. Springer. 35–48.

Lefebvre, A., and Lecroq, T. 2002. Compror: on-line lossless data compression with a factor oracle. *Information Processing Letters* 83(1):1–6.

Lefebvre, A.; Lecroq, T.; and Alexandre, J. 2003. An improved algorithm for finding longest repeats with a modified factor oracle. *Journal of Automata, Languages and Combinatorics* 8(4):647–657.

Pachet, F., and Roy, P. 2011. Markov constraints: steerable generation of markov sequences. *Constraints* 16(2):148–172.

Roy, P., and Pachet, F. 2013. Enforcing meter in finite-length markov sequences. In *AAAI*.

Surges, G., and Dubnov, S. 2013. Feature selection and composition using pyoracle. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Wainwright, M. J., and Jordan, M. I. 2008. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning* 1(1-2):1–305.