Improving and Adapting Finite State Transducer Methods for Musical Accompaniment

Jonathan P. Forsyth
Music and Audio Research Lab (MARL)
New York University
jpf211@nyu.edu

Rachel M. Bittner
Music and Audio Research Lab (MARL)
New York University
rachel.bittner@nyu.edu

Michael Musick
Music and Audio Research Lab (MARL)
New York University
musick@nyu.edu

Juan P. Bello
Music and Audio Research Lab (MARL)
New York University
jpbello@nyu.edu

ABSTRACT

A common area of research within music technology is the automatic generation of musical accompaniment. Of particular interest is the generation of harmonic accompaniment to melody. While there are many approaches to solving this problem, the use of various types of finite state machines is popular. One such finite state machine, the finite state transducer (FST), has been used in a few accompaniment generation systems. Although the FST is widely used in speech recognition, its application in accompaniment generation systems has been relatively unsophisticated. In this paper, we introduce an improved approach to generating harmonic accompaniment to melody based on techniques used in speech recognition. In addition, we describe how we extend this method to the generation of rhythmic accompaniment. Finally, we discuss the integration of these offline accompaniment systems into a real-time system.

1. INTRODUCTION

Composers, performers, and researchers have been developing a wide variety of interactive music systems for over 40 years. Some systems are designed for use in a live improvisational context, while others are organized around a "score-following" paradigm in which a system attempts to align a live performance with a pre-existing score. A few of these systems are specifically designed to generate some form of accompaniment in real time. Other, closely related research has focused on the development of offline solutions to the problem of generating musical accompaniment, usually in the form of harmonic accompaniment to melody. Some of these automatic accompaniment generation systems [1,2] use a particular type of finite state machine, a *finite state transducer* (FST), to generate harmonic accompaniment. While FSTs are well-suited to the task of accompaniment generation, these systems use a fairly simple transducer topology, and are designed for offline use.

In this paper, we propose an improved FST topology inspired by techniques used in speech recognition. In addition, we describe how this approach is extended to generate rhythmic as well as harmonic accompaniment. We also discuss adapting our approach for a real-time context, in the form of a collaborative music-creation installation entitled *The Harmonically Ecosystemic Machine; Sonic Space No. 7*, and describe the various challenges and compromises involved in implementing the new FST topology and adapting the system to a real-time context.

The remainder of this paper is organized as follows. In the next section, we provide background on relevant interactive music and automatic accompaniment generation systems. In section 3, we provide background information on finite state transducers. In section 4, we discuss our current approach to harmonic accompaniment generation and an approach to rhythmic accompaniment generation, and in section 5, we describe the application of these systems to a real-time context. Finally, in section 6 we discuss our conclusions and provide directions for future work.

2. RELATED WORK

In this section, we provide some of the context for the current work within the fields of interactive music systems and offline accompaniment generation systems.

2.1 Interactive Music Systems

Interactive music systems, defined by Rowe as systems "whose behavior changes in response to musical input" [3], include a great many different types of systems, and cover a variety of musical applications. A number of systems, such as the *Music Plus One* [4] and *Antescofo* systems [5], are *score-followers*,

i.e. systems designed to align a human performance with a pre-defined score and generate appropriate musical material from the score.

Some systems make use of a score-following paradigm within a system for generating improvisations. For example, the system described by Ramalho et al. [6] uses a framework of artificially intelligent software agents to generate idiomatic jazz bass lines over a specified chord progression. These agents make use of melodic phrases played by human musicians. Similarly, the *Cyber-João* system [7] assembles guitar accompaniment in the Bossa Nova style by using a database of transcriptions of rhythmic patterns played by guitarist João Gilberto on various recordings.

By contrast, many systems are intended for use in purely improvisational settings, with no pre-defined score. An early, and important, example is George Lewis' *Voyager*, which is designed to generate improvisations independently as well as in response to a human performer [8]. Voyager is designed specifically to generate musical content in Lewis' distinctive aesthetic. Robert Rowe's *Cypher* attempts to achieve more flexibility by exposing many of the system's musical parameters to the user [3].

Other systems achieve aesthetic flexibility using a data-driven approach. For example, the *OMax* system models various aspects of musical style using the *factor oracle*, a particular type of finite state machine used to model sequential data [9]. These performance models, which can be learned offline as well as in real-time, are used to generate improvisations in the style of a particular performer. Pachet's *Continuator* shares this goal of automatically generating musical phrases in a particular style [10]. The *Continuator* is also similar to *OMax* in that it models sequences of musical features using a finite state machine, in this case a *variable length Markov chain*.

2.2 Offline Accompaniment Generation Systems

In addition to the real-time systems described above, there are a number of offline accompaniment generation tools, many of which generate harmonic accompaniment to a given melody. Harmonizing melodies in the style of J.S. Bach is particularly popular. For example, Allan and Williams describe a system that uses *hidden Markov models* (HMMs) to model the relationship between melody notes and chords [11]. In this case, melody notes are considered observations and chords the hidden states of the HMM, which is trained on a corpus of Bach chorales.

The *MySong* system [12], intended to be used by novice musicians, employs an HMM-based approach similar to that of [11]. However, it is designed specifically to generate harmonizations of vocal melodies in more popular music styles, and is thus trained on a database of pop, rock, R&B, jazz, and country lead sheets. The system described by Chuan and Chew [13] is also geared towards novices. However, this system uses a combination of machine learning and rule-based approaches.

The system described by Buys and van der Merwe [1] attempts to reproduce Bach's harmonization and voice-leading style. Here, the authors use a weighted (probabilistic) finite state transducer to map melody notes to chords, and a weighted finite state automaton to model chord sequences. Other weighted FSTs are used to model the movement of the inner and bass voices. In an earlier paper, we developed a system using a similar approach [2]. Although this system is also trained on a dataset of Bach chorales, the goal was to reproduce the general accompaniment style inherent in the training data, and not to reproduce Bach's voice-leading techniques. This approach serves as the basis of the architecture described in section 4.

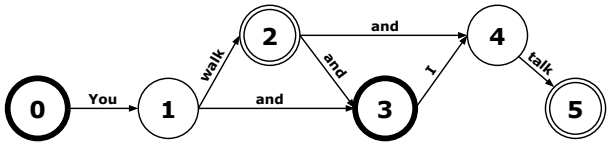


Figure 1: A simple FSA representing a set of grammatical sentences containing the words "You", "walk", "and", "I", "talk".

3. FINITE STATE AUTOMATA AND TRANSDUCERS

In this section, we provide a general overview of two types of finite state machines, the finite state automaton and finite state transducer. The reader is referred to [14, 15] for a more detailed description of finite state machines, and to [16] for more information about the application of finite state machines in speech recognition.

Finite state machines are graphical models which are typically represented as directed graphs consisting of nodes (states), and edges (transitions). *Finite state automata* (FSAs) are a type of finite state machine for which each edge has a single transition label. An FSA is defined by a finite *alphabet* Σ, a set of *initial states* and *final states*, and a set of *transitions* between states. The alphabet defines the set of possible values a transition label can have, and is augmented with a "null" symbol ϵ. The initial and final states determine the starting and ending states of the machine, and the transitions determine what states can be accessed from a given state. The combination of the alphabet, initial and final states, and transitions defines the set of possible *paths* accepted by an FSA. Any path (sequence of symbols from Σ) *accepted* by an FSA begins at an initial state, proceeds symbol by symbol through the graph structure based on valid transitions given the current state and symbol, and terminates at a final state.

In the case of a *weighted* FSA, each transition has an associated weight, which can represent values such as the probability that the associated transition occurs. In this case, each valid path through the FSA will have an associated weight. For example, if the weights represent transition probabilities, the total weight of a valid path represents the probability of the symbol sequence defined by that path.

As an example, Figure 1 depicts an unweighted FSA defining a simple language model. Its initial states are depicted

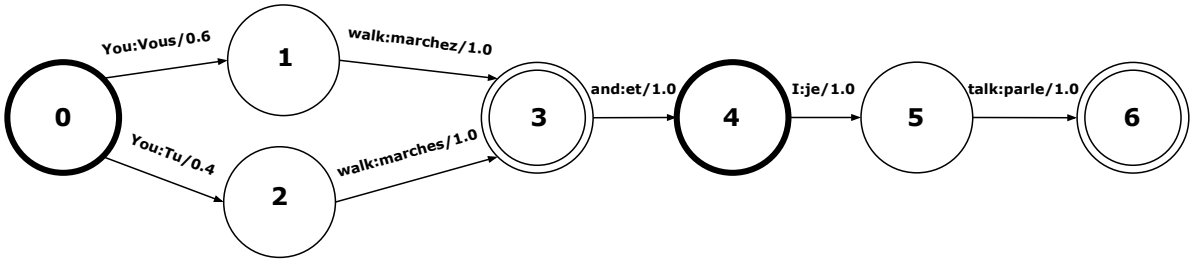


Figure 2: An FST that translates the English sentence "You walk and I talk" into French.

as bold circles (states 0 and 3), and its final states are depicted as double circles (states 2 and 5). The alphabet for this FSA is $\Sigma = \{$You, walk, and, I, talk$\}$. Strings that would be accepted this FSA are: "You walk and I talk" (with the associated path passing through states $0, 1, 2, 3, 4, 5$), "You walk and talk" (states $0, 1, 2, 4, 5$), "You and I talk" (states $0, 1, 3, 4, 5$), "You walk" (states $0, 1, 2$), and "I talk" (states $3, 4, 5$).

Note that the string "You and I" (states $0, 1, 3, 4$) would not be accepted, because state 4 is not final. Similarly, given the string "You and you talk," the FSA would successfully parse the substring "You and" (states $0, 1, 3$), but, because there is no transition from state 3 labelled "you," the string would not be accepted.

Finite state transducers (FSTs) are similar to FSAs, except that their edges are labeled with an input/output pair. They are defined by finite input and output alphabets, a set of initial and final states, and a set of transitions between states. Like FSAs, FSTs can be weighted. For a given *input string* (sequence of symbols from the input alphabet), an FST will produce a set of possible *output strings* (sequence of symbols from the output alphabet).

An illustrative example of a weighted FST is given in Figure 2, which "translates" simple English sentences into French sentences. As with the FSA example, the initial states 0 and 4 are shown in bold and the final states 3 and 6 are shown as double circles. The input alphabet for this FST is $\{$You, walk, and, I, talk$\}$, and the output alphabet is $\{$Vous, Tu, marchez, marches, et, je, parle$\}$. Transition labels are of the form "input:output/weight". When the weights are probabilities, the weight of a complete path can be computed as the product of the probabilities of the transitions. Given the input string "You walk", this FST would return the two strings "Vous marchez" with weight 0.6, and "Tu marches" with weight 0.4. Given the input string "I talk", this FST would return "je parle" with weight 1.0. However, given the input string "walk and", the FST would return nothing; there is no complete path for the input string "walk and", since valid paths must start at an initial state and end at a final state.

There are a number of operations that can be applied to FSAs, FSTs, and combinations thereof [17]. The most important of these operations for the purposes of this paper is *composition*, which combines two machines into a single machine. The composition of the FSTs $\mathcal{T}_1$ and $\mathcal{T}_2$ into the FST $\mathcal{T}$ is denoted $\mathcal{T} = \mathcal{T}_1 \circ \mathcal{T}_2$, and requires that the output alphabet of $\mathcal{T}_1$ is equal to the input alphabet of $\mathcal{T}_2$ (note that, in the case of FSAs, the input and output alphabets are identical). For example, the FSA in Figure 1 could be composed with the FST in Figure 2. Through the use of composition, separate machines representing components of a complex system can be combined to model the system as a whole.

4. AUTOMATIC HARMONIC ACCOMPANIMENT GENERATION

In this section, we describe an approach to generating harmonic accompaniment using an improved FST topology. In addition, we detail a new approach to generating rhythmic accompaniment based on our methods for harmonic accompaniment generation. We also discuss the process of adapting these accompaniment techniques for a real-time context.

The implementations of the systems described in the following sections make extensive use of the excellent OpenFst [18] and OpenGrm [19] software libraries.

4.1 General Framework

Our approach to harmonic accompaniment models harmony as the output of an input melody sequence. For an input melody sequence $\mathbf{m} = [m_1, m_2, \ldots m_n]$, the goal is to find the most likely sequence of chords $\hat{\mathbf{c}} = [c_1, c_2, \ldots c_m]$. We can describe the problem of finding $\hat{c}$ as:

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c} \in \Sigma^*} \Pr\left[\mathbf{c} \mid \mathbf{m}\right] = \arg\max_{\mathbf{c} \in \Sigma^*} \Pr\left[\mathbf{m} \mid \mathbf{c}\right] \cdot \Pr[\mathbf{c}] \quad (1)$$

where Σ^* is the set of possible chord sequences. The formulation in Equation (1) allows us to develop separate models for $\Pr[\mathbf{m} \mid \mathbf{c}]$ and $\Pr[\mathbf{c}]$. The melody to chord model, $\Pr[\mathbf{m}|\mathbf{c}]$, can be estimated by learning the distribution of melody-chord pairs from a dataset, while $\Pr[\mathbf{c}]$ can be estimated by learning an n-gram [20] model trained on a dataset of chord sequences. We can model $\Pr[\mathbf{m}|\mathbf{c}]$ using an FST, $\mathcal{L}$, and $\Pr[\mathbf{c}]$ using an n-gram, $\mathcal{G}$. $\mathcal{L}$ and $\mathcal{G}$ can then be composed into a single machine, $\mathcal{T} = \mathcal{L} \circ \mathcal{G}$. Our proposed framework is similar in structure to the framework used in speech recognition, with melody notes representing "phonemes" and chords representing "words".

Given a trained model $\mathcal{T}$ and an input melody sequence $\mathbf{m}$, the most likely chords are found by first converting $\mathbf{m}$ into

a *linear FSA* (i.e. an FSA that accepts only one string), $\mathcal{M}$. The composition of the linear FSA and the full model is then computed as $\mathcal{C} = \mathcal{M} \circ \mathcal{T}$. The optimal transcription of the observation sequence can be found by solving for the shortest path (the path with the smallest weight) through $\mathcal{C}$. Note that, in this case, the weights represent negative log probabilities, and thus the shortest path through a machine corresponds to the path with the highest probability.

The shortest path through $\mathcal{C}$ would normally be considered to be the "optimal" harmonization. However, the concept of having a "correct" answer is one of the fundamental places where this task differs from the speech recognition framework. In most cases in speech recognition, there is a single correct transcription, and thus the goal of choosing the *most likely* path through a machine is optimal.

For harmonizations, there are always multiple correct answers; thus, choosing the shortest path through a machine may not be the most desirable choice, as it will choose the most "common" progression. Unfortunately, the most common chord progression is typically a rather simple and uninteresting accompaniment. As an alternative, we can generate random paths through $\mathcal{C}$ by sampling the paths while treating the weights as negative log probabilities, or by sampling the paths uniformly. This will always produce "correct", if not "optimal", accompaniments, because the accepted paths through the FST $\mathcal{C}$ are all valid matches to the given melody.

4.2 Training $\mathcal{L}$ and $\mathcal{G}$

There are a number of different possible FST topologies for $\mathcal{L}$. For example, in [2], the authors use a single-state FST in which each pitch class is mapped to each chord type with the associated probability of their co-occurrence as the weight. Unfortunately, this strategy fails to consider melodic context, and requires that every melodic note receive a chord. Instead of the single-state topology, we use a more flexible topology inspired by FSTs used in speech recognition to map sequences of phonemes to words [16]. This allows any number of notes to be assigned to a single chord.

The FSTs $\mathcal{L}$ and $\mathcal{G}$ are trained using a corpus of aligned symbolic chords and melodies. Each of these chord/melody sequences is divided such that each segment contains only one chord, but may contain multiple melody notes [1] . These segments are then converted to symbolic mappings between the melody sequence and the chord. For simplicity, we represent melody notes using their pitch class, and disregard their octave, metrical position, and duration. For example, Figure 3 shows a melody and an aligned accompaniment; the training data for this sequence would be the pairs: $[g, g, d, d : G]$, $[e, e : C]$, $[d : G]$, $[c, c : C]$, $[b, b : G]$, $[a, a : D7]$, and $[g : G]$.

The weighted FST $\mathcal{L}$ is created by combining these segments. Each segment is first written as a linear FST, and each such segment is given equal weight. These linear FSTs are combined by creating a global initial state and final state,

[1] If a chord duration is shorter than its corresponding note, the note is split and repeated so that at most one chord corresponds to a single melody note.

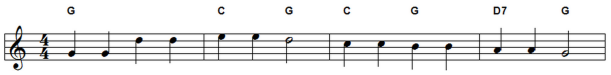


Figure 3: Example of a symbolic training sequence.

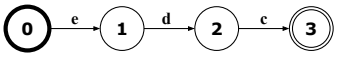
which is connected respectively to the first and last state of each linear FST. An ϵ transition is added to connect the final and initial state. An example of such an $\mathcal{L}$ is shown in Figure 4b. Finally $\mathcal{L}$ is minimized and then determinized, which reduces the FST to a minimally equivalent form, and combines the weights of repeated sequences.

The n-gram model $\mathcal{G}$ is trained using all length-n chord sequences that occur in the training data. For the music in Figure 3, the training samples would be all ordered length-n subsequences of the sequence $[G, C, G, C, G, D7, G]$. In our system we use $n = 3$, but this choice is flexible and the best n may vary with the musical style. This set of length-n sequences is used to determine a probability distribution across all n-grams, which is represented as a weighted FSA.

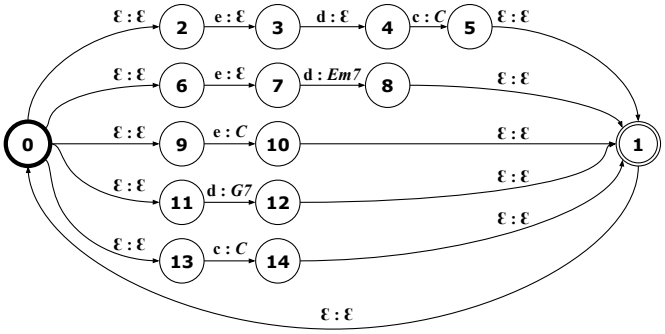
4.3 A Simple Example

Figure 4 presents an example of a complete pipeline. Note first, that the input alphabet is Σ_i ={c, d, e}, and the output alphabet is Σ_o ={C, Em7, G7}. Both alphabets are augmented with the symbol ϵ, which represents an empty transition. The FSTs $\mathcal{L}$ and $\mathcal{G}$ in this system were "trained" on different corpuses - the corpus for $\mathcal{L}$ containing 5 melody-chord sequences, and the corpus for $\mathcal{G}$ containing transitions between the chords C and G7.

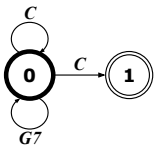
Figure 4a shows a linear FSA $\mathcal{M}$ corresponding to the input melody sequence shown in Figure 5. Figure 4b shows a trained FST $\mathcal{L}$, which gives mappings for input sequences. Note that in this example, for simplicity, $\mathcal{L}$ is not weighted, but if it were, each of the 5 forward paths would have equal weight. Figure 4c gives a simple (again unweighted) n-gram model for the output chord sequences, which in this case only allows combinations of C and $G7$ which end with a C. When $\mathcal{M}$ is composed with $\mathcal{L}$, there are 3 accepted paths through the states: $[0, 2, 3, 4, 5, 1]$, $[0, 6, 7, 8, 1, 0, 13, 14, 1]$, and $[0, 9, 10, 1, 0, 11, 12, 1, 0, 13, 14, 1]$. However, $\mathcal{M}$ is composed with the *composition* of $\mathcal{L}$ and $\mathcal{G}$. The composition of $\mathcal{L}$ with $\mathcal{G}$ adds an additional constraint on the accepted paths, and in this example removes the path $[0, 6, 7, 8, 1, 0, 13, 14, 1]$ that was possible when $\mathcal{M}$ was composed with $\mathcal{L}$. This leads to the final output FST $\mathcal{C} = \mathcal{M} \circ (\mathcal{L} \circ \mathcal{G})$ in Figure 4d. Note that C is not a single path, but a transducer itself, where each accepted path gives a valid output sequence for the input sequence [e d c]. In this example, the possible accompaniments for the melody $\mathcal{M}$ are the single chord C for all three notes (the upper path in Figure 4d, and the red chords in Figure 5), or the chords C, $G7$, C (the lower path in Figure 4d, and the blue chords in Figure 5). A single output could be chosen by selecting randomly from these two outputs; if given weights, this selection chosen randomly based on the distribution determined by the weights.



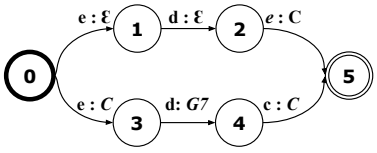
(a) Input melody sequence $\mathcal{M}$.



(b) Melody to chord FST $\mathcal{L}$.



(c) N-gram $\mathcal{G}$.



(d) Output chord FST $\mathcal{C}$.

Figure 4: A simple example illustrating the accompaniment generation system.

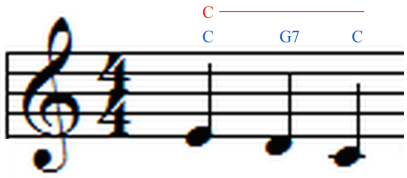


Figure 5: Music notation version of the FSA in Figure 4a, and the two possible accompaniments given by the FST in Figure 4d

4.4 Disambiguation of Conflicting Mappings

Using this FST topology introduces a problem often encountered in speech recognition. The problem manifests itself when we have in our dataset two identical phoneme sequences that map to different words. For example, "there" and "their" have the same pronunciation, and thus are represented by the same sequences of phonemes, but clearly map to different words. If we construct an FST without accounting for these cases, the transducer cannot be determinized, and consequently, cannot be minimized. Determinization and minimization are necessary steps in the construction of compact and efficient machines.

The same problem exists at a much larger scale in the context of melody sequences. For any given melody sequence, there are usually many possible chords that can correspond to it. In order to resolve this problem, we again borrow from a technique used in speech recognition and introduce disambiguation symbols (typically $\#0, \#1, \#2, \ldots$). If we have a melody sequence that maps to a number of different chords, we generate a unique melody sequence for each chord by appending a unique disambiguation symbol to the end of the melody sequence.

For example, the melody sequence $[c, e, g]$ may map to a C major chord. However, this sequence may also map to an Amin7 chord. In order to resolve this problem, we disambiguate the two melodies by mapping the sequence $[c, e, g, \#0]$ to $Cmaj$ and the sequence $[c, e, g, \#1]$ to $Amin7$. If we properly disambiguate all melodies using this method, the resulting FST can be determinized and minimized. Note that after these steps, we replace all the disambiguation symbols in the FST with the null symbol (ϵ).

4.5 Extension to Rhythm

We can apply the general approach described above to the problem of generating rhythmic accompaniment. In this case, given a particular rhythmic pattern the goal is to automatically generate another rhythmic pattern that can serve as an acceptable accompaniment to the input pattern.

We formulate the problem as follows. Given an input rhythm sequence $\mathbf{s} = [s_1, s_2, \ldots s_n]$, we wish to find the optimal sequence of accompanying rhythms $\hat{\mathbf{r}} = [r_1, r_2, \ldots r_m]$. As above, we estimate $\hat{\mathbf{r}}$ as:

$$\hat{\mathbf{r}} = \arg\max_{\mathbf{r} \in \Sigma^*} \Pr[\mathbf{r} \mid \mathbf{s}] = \arg\max_{\mathbf{r} \in \Sigma^*} \Pr[\mathbf{s} \mid \mathbf{r}] \cdot \Pr[\mathbf{r}] \quad (2)$$

where Σ is the alphabet of quantized rhythm values. For example, we can represent the rhythms using quantized inter-onset beat values.

In similar fashion to our harmonic generation method, we model $\Pr[\mathbf{s}|\mathbf{r}]$ with an FST, $\mathcal{L}$, and $\Pr[\mathbf{r}]$ with an n-gram, $\mathcal{G}$. We train both $\mathcal{L}$ and $\mathcal{G}$ on a database of pairs of quantized rhythm sequences using a process nearly identical to the one described in Section 4.2; we use the method described in Section 4.1 to compute the accompaniment pattern. Note that here we are creating a separate model for rhythmic accompaniment that could be used in conjunction with the harmonic accompaniment model discussed above or any other generative model.

5. ACCOMPANIMENT GENERATION IN REAL-TIME

Both the harmonic and rhythmic accompaniment generation systems described in the previous sections were incorporated into an installation project, *The Harmonically Ecosystemic Machine; Sonic Space No. 7*. In this section, we provide a brief description of this installation and the challenges involved in adapting our systems to a real-time context. A more

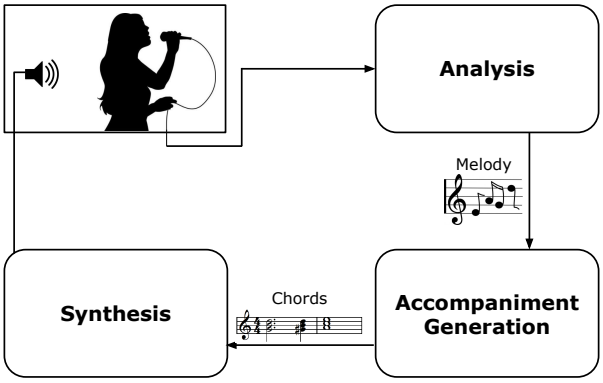


Figure 6: Top-level system diagram for *The Harmonically Ecosystemic Machine; Sonic Space No. 7* installation.

complete description of this project can be found in [21]. In addition, more information, including recorded audio examples and code, can be found at `http://steinhardt.nyu.edu/marl/research/sonic_spaces`.

5.1 System Design

The Harmonically Ecosystemic Machine; Sonic Space No. 7 is an interactive installation in which participants are encouraged to sing or use instruments placed in the physical space. The installation is not intended to directly provide accompaniment, but rather to reflect and reinterpret the musical material produced by the participants in a particular harmonic and rhythmic context. In addition, the system contains a feedback loop, and thus responds to the musical material produced by both the human participants and by the system itself.

A diagram illustrating the overall system design is shown in Figure 6. As seen in the figure, the system consists of an analysis, accompaniment generation, and synthesis module. The system's analysis module analyzes the music signals in the room in order to estimate the key, tempo, melodies, and rhythms being performed. The module then assembles this information into sequences of melodies and rhythms. In addition, the analysis module records separate musical events, storing a number of features, such as pitch, with each recording.

The melodic and rhythmic sequences created by the analysis module are sent to the accompaniment generation module. As shown in Figure 7, this module consists of two submodules: one to generate harmonic accompaniment and a second to generate rhythmic accompaniment using the methods detailed in Section 4. The harmonic accompaniment model is trained with data from the Rock Corpus dataset [22], a set of transcriptions of the melodies and harmonies of 200 pop, rock, and R&B songs; all songs are transposed to the key of C major before training the FST. The rhythmic accompaniment model was trained with rhythms extracted from compositions by Mozart included in the music21 Python toolkit [23]. These training datasets were chosen so that the generated harmonies and rhythms were interesting, but also familiar to users of the

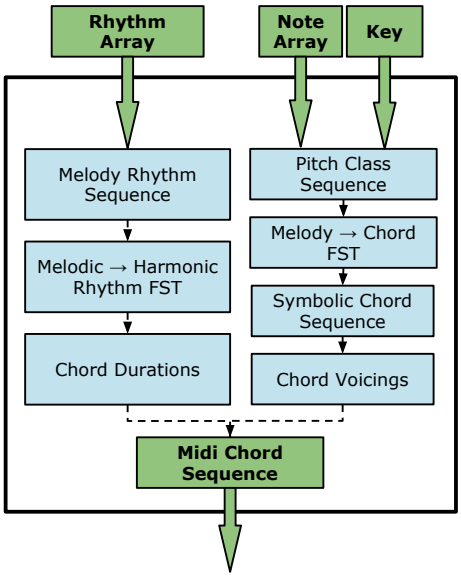


Figure 7: Schematic of the accompaniment generation module.

system.

Melodic and rhythmic sequences from the analysis module are sent separately to each sub-module. The melodic sequence is transposed to C, and the resulting sequence of pitch classes is used as input to the melody-to-chord FST. The FST outputs a sequence of chords, each of which is transposed to the original key, and converted from a chord symbol to a fully-voiced chord. Similarly, the input rhythmic sequence is used to generate an accompaniment rhythmic sequence. Finally, the outputs of the two sub-modules combined to form a single MIDI chord sequence.

The output of the accompaniment generation module is then sent to the synthesis module. This module assembles the various pre-recorded musical events according to their pitch content in order to reproduce the generated harmonic sequences; these reconstructed harmonies are played back according to the generated rhythmic sequences and the estimated global tempo.

5.2 Adapting the System to Real-time

Incorporating our approach to accompaniment generation within the installation project involves a number of compromises. While the overall efficiency is less important in an offline system, latency becomes a primary concern in a real-time context. A number of factors can impact the performance of the system. In particular, the size of an FST can effect its efficiency; operations such as composition and computing the shortest path take longer with larger FSTs. Because this problem is particularly acute in the case of the harmonic accompaniment FST, we employ a number of strategies to keep it as lightweight and compact as possible.

One such strategy involves how we pre-process the training data. If we train our melody-to-chord FST $\mathcal{L}$ (see Sec-

tion 4) without performing any transpositions on the data, the resulting model will be unable to produce proper harmonizations for melodies in keys not present in the training data. Solutions to this problem include key-normalizing the dataset (i.e., transposing all training examples to the key of C major), or transposing the training data to all 12 keys. The former approach requires estimation of the key of a melody. Given the errors inherent in any key estimation algorithm, a model trained using the latter approach would likely generate more accurate harmonizations. However, the model produced using the latter approach will be significantly larger than a model trained on key-normalized data. Therefore, for the purposes of the installation, the improved efficiency of the key-normalization approach makes up for any errors introduced in the key estimation process.

In addition, we use an n-gram order of 3 to limit the size of the chord sequence model $\mathcal{G}$, although this decision likely has less of an impact on the aesthetic quality of the results, as there is currently no accepted optimal value for the n-gram order in this context. These strategies reduce the size of the full harmonic accompaniment model $\mathcal{L} \circ \mathcal{G}$.

Another concern arises from the latency incurred by the length of the sequences used as input to the accompaniment generation module. Recall that in order to generate the harmonic accompaniment to a melody sequence, we construct a linear FSA, $\mathcal{M}$, from the melody sequence, and compose this FSA with the accompaniment model as $\mathcal{C} = \mathcal{M} \circ (\mathcal{L} \circ \mathcal{G})$, as described in Section 4. Just as reducing the size of $\mathcal{L} \circ \mathcal{G}$ improves the efficiency of the system, so does reducing the size of $\mathcal{M}$. In other words, using shorter melody sequences results in improved efficiency with regards to computing $\mathcal{C}$. Reducing the length of melody sequences also increases the responsiveness of the system merely from the fact that sending shorter melody sequences to the harmonic generation submodule implies a shorter time interval between the completion of a melody and the generation of the accompaniment.

However, longer melodic contexts will in general produce higher quality harmonizations, and thus there is a clear tradeoff between latency and harmonic accompaniment quality. Because the aesthetic goals of *The Harmonically Ecosystemic Machine* installation did not require the system to immediately respond to musical input, we were able to err on the side of providing a relatively long melodic context (eight beats).

6. CONCLUSIONS AND FUTURE DIRECTIONS

We have presented a flexible, data-driven, FST-based system for automatically generating musical accompaniment to a given melody. The FST topology is an extension of earlier approaches, and is inspired by techniques used widely in speech recognition. We have also shown how this approach can be modified to generate rhythmic accompaniment. Additionally, we have presented an application that uses this algorithm in a real-time setting, and have discussed how the system was adapted to work in real-time. The presented accompaniment-generation framework can be used for virtually any musical style, is extendable to other musical applications, and is easily incorporated into a real-time system.

There are a number of aspects of the system that we are currently working to improve. One known weakness of the current system is its lack of explicit rhythmic encoding, which is crucial for music. We are currently exploring adjusting the weights on the transitions of the melody-to-chord $\mathcal{L}$ machine to account for the metrical position and duration of each melody note.

Another weakness of the current design occurs when an input melody is very different from combinations of those present in the training data. In this case, there will be very few options available for the harmonic accompaniment. Thus, as a second improvement, we plan to incorporate the use of an *edit transducer*, which may allow the machine to better account for partial matches between training set and input melody sequences.

The original approach described is built for generating harmonic accompaniment for a given melody, but, as we have shown, the same framework can be applied to the problem of generating rhythmic accompaniment patterns. This approach can easily be applied to many other settings. In particular, our formulation can be used to generate a symbolic musical sequence given any other symbolic music sequence, as long as sufficient training data is available. For example, the inverse problem - to generate melodies for a given musical accompaniment - is an obvious extension of the framework. Similarly, it can also be used to generate a bass line for a given chord sequence, or for a given melody. Our future work includes exploring some of these possibilities.

Encouraged by the success of *The Harmonically Ecosystemic Machine; Sonic Space No. 7* project, we are interested in applying our accompaniment generation framework to other real-time systems. We have a particular interest in applying our accompaniment generation framework to more "traditional" interactive music systems.

The flexible nature of the FST-based approach allows us to explore, to address problems, and to continue to extend the system in a variety of directions with relative ease.

7. REFERENCES

[1] J. Buys and B. van der Merwe, "Chorale Harmonization with Weighted Finite-State Transducers," in *Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa (PRASA)*, 2012, pp. 95–101.

[2] J. P. Forsyth and J. P. Bello, "Generating Musical Accompaniment Using Finite State Transducers," in *Proceedings of the 16th Int. Conf. on Digital Audio Effects (DAFx). Maynooth, Ireland (Sept 2013)*, 2013.

[3] R. Rowe, *Interactive Music Systems: Machine Listening and Composing*. The MIT Press, 1992.

[4] C. Raphael, "Music Plus One and Machine Learning," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 21–28.

[5] A. Cont, "ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music." in *Proceedings of the International Computer Music Conference (ICMC)*, 2008, pp. 33–40.

[6] G. L. Ramalho, P.-Y. Rolland, and J.-G. Ganascia, "An Artificially Intelligent Jazz Performer," *Journal of New Music Research*, vol. 28, no. 2, pp. 105–129, 1999.

[7] M. Dahia, H. Santana, E. Trajano, G. Ramalho, C. Sandroni, and G. Cabral, "Using Patterns to Generate Rhythmic Accompaniment for Guitar," *CEP*, vol. 50732, p. 970, 2004.

[8] G. E. Lewis, "Too Many Notes: Computers, Complexity and Culture in Voyager," *Leonardo Music Journal*, vol. 10, pp. 33–39, Dec. 2000.

[9] G. Assayag, G. Bloch, M. Chemillier, A. Cont, and S. Dubnov, "Omax Brothers: A Dynamic Typology of Agents for Improvization Learning," in *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia*. ACM, 2006, pp. 125–132.

[10] F. Pachet, "The Continuator: Musical Interaction with Style," *Journal of New Music Research*, vol. 32, no. 3, pp. 333–341, 2003.

[11] M. Allan and C. K. Williams, "Harmonising Chorales by Probabilistic Inference," *Advances in Neural Information Processing Systems*, vol. 17, pp. 25–32, 2005.

[12] I. Simon, D. Morris, and S. Basu, "MySong: Automatic Accompaniment Generation for Vocal Melodies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 725–734.

[13] C.-H. Chuan and E. Chew, "A Hybrid System for Automatic Generation of Style-Specific Accompaniment," in *Proceedings of the International Joint Workshop on Computational Creativity*, 2007, pp. 57–64.

[14] E. Vidal, F. Thollard, C. De La Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic Finite-State Machines – Part I," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 7, pp. 1013–1025, 2005.

[15] ——, "Probabilistic Finite-State Machines – Part II," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 7, pp. 1026–1039, 2005.

[16] M. Mohri, F. Pereira, and M. Riley, "Speech Recognition with Weighted Finite-State Transducers," in *Springer Handbook of Speech Processing*. Springer, 2008, pp. 559–584.

[17] M. Mohri, "Weighted Automata Algorithms," in *Handbook of Weighted Automata*. Springer, 2009, pp. 213–254.

[18] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A General and Efficient Weighted Finite-State Transducer Library," in *Implementation and Application of Automata*. Springer, 2007, pp. 11–23.

[19] B. Roark, R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai, "The OpenGrm Open-Source Finite-State Grammar Software Libraries," in *Proceedings of the ACL 2012 System Demonstrations*. Association for Computational Linguistics, 2012, pp. 61–66.

[20] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram Models of Natural Language," *Computational Linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[21] M. Musick, J. P. Forsyth, and R. Bittner, "Building a Harmonically Ecosystemic Machine: Combining Sonic Ecosystems with Models of Contemporary Harmonic Language," in *Proceedings of the International Computer Music Conference (ICMC)*, 2015.

[22] T. De Clercq and D. Temperley, "A Corpus Analysis of Rock Harmony," *Popular Music*, vol. 30, no. 01, pp. 47–70, 2011.

[23] M. S. Cuthbert and C. Ariza, "music21: A Toolkit for Computer-aided Musicology and Symbolic Music Data," in *Proceedings of the International Symposium on Music Information Retrieval*, vol. 11, 2010, pp. 637–42.