Improving and Adapting Finite State Transducer Methods for Musical Accompaniment

Jonathan P. Forsyth, Rachel M. Bittner, Michael Musick, & Juan P. Bello
Music and Audio Research Lab (MARL)
New York University
New York, NY
introduction

given an input melody,
generate harmonic accompaniment

data-driven approaches

finite state machines are a natural fit for solving this problem, and have been used extensively in natural language processing
outline

introduction to finite state machines

general approach to generating musical accompaniment

application of this approach
finite state machines

used in natural language processing, bioinformatics, computer vision

directed graphs (nodes, edges)
edges labelled with symbols
edges and nodes can have weights (e.g. probabilities)
finite state machines

make it relatively easy to break down a complex problem into simpler sub-problems

allow use of a purely data-driven approach

can also incorporate higher-level knowledge into system

require relatively little training data
finite state automaton

FSA defined by the 5-tuple:

\[ \langle \Sigma, Q, I, F, \delta \rangle \]

(sometimes referred to as an “acceptor”)

Monday, February 15, 2016
finite state transducer

FST defined by the 6-tuple:

\[ (\Sigma, \Delta, Q, I, F, \delta) \]
finite state operations

composition
determinization
minimization
shortest path
generating accompaniment

approach to accompaniment generation borrowed from speech recognition

speech recognition problem:

find the most likely sequence of words given a sequence of phonemes
generating accompaniment

general framework:

\[ \hat{c} = \arg \max_{c \in \Sigma^*} \Pr [c | m] = \arg \max_{c \in \Sigma^*} \Pr [m | c] \cdot \Pr [c] \]

\(m\): input melodic sequence
\(\hat{c}\): most likely sequence of accompaniment chords
generating accompaniment

\[ \hat{c} = \arg \max_{c \in \Sigma^*} \Pr[c | m] = \arg \max_{c \in \Sigma^*} \Pr[m | c] \cdot \Pr[c] \]

model components separately:

- FST (L) to model \( \Pr[m | c] \)
- FSA (G) to model \( \Pr[c] \)
generating accompaniment

input alphabet ($\Sigma_i$): pitch classes

output alphabet ($\Sigma_o$): chord symbols or quantized chroma vectors\[1\]

training data: pairs of sequences of melodic symbols and sequences of harmonic symbols

training FST

example training sequence
training $L$ FST

generates the training sequences:

\begin{align*}
g, g, d, d & \rightarrow G \\
e, e & \rightarrow C \\
d & \rightarrow G \\
c, c & \rightarrow C \\
b, b & \rightarrow G \\
a, a & \rightarrow D7 \\
g & \rightarrow G
\end{align*}
training L FST
training $\mathcal{L}$ FST
same melodic sequence may map to different chords

(e.g. [c,e,g] may map to C or Amin7)
same melodic sequence may map to different chords
(e.g. \([c,e,g]\) may map to \(C\) or \(A\text{min7}\))

analogous problem in speech recognition:
same set of phonemes map to “red” and “read”
training $\mathcal{L}$ FST

an FST with conflicting mappings cannot be determinized, which is required for minimization
training $\mathcal{L}$ FST

solution:

append different disambiguation symbol (typically, ‘#0’, ‘#1’, etc.) to each sequence that maps to more than one chord
training $\mathcal{L}$ FST

e.g., our training data contains pairs $[c, e, g : C]$ and $[c, e, g : Amin7]$

add disambiguation symbols to get the pairs

$[c, e, g, #0 : C]$ and $[c, e, g, #1 : Amin7]$
training $\mathcal{L}$ FST

after training full model, replace disambiguation symbols with $\varepsilon$
training G FSA

use an n-gram trained on all length-n chord sequences in training data

n-grams model sequential data with context of n-1 symbols

n-gram represented as an FSA (G)
full model

combine the individual models:

\[ \mathcal{T} = \pi_\epsilon(\min(\det(\tilde{L} \circ G))) \]
full model

combine the individual models:

$$\mathcal{T} = \pi_\epsilon(\min(\det(\tilde{L} \circ G)))$$
full model

combine the individual models:

$$\mathcal{T} = \pi_\varepsilon (\text{min}(\text{det}(\hat{L} \circ G)))$$
full model

combine the individual models:

\[ \mathcal{T} = \pi_\epsilon(min(det(\tilde{\mathcal{L}} \circ \mathcal{G}))) \]
full model

combine the individual models:

\[ \mathcal{T} = \pi_\epsilon(\min(\det(\mathcal{L} \circ \mathcal{G}))) \]
full model

combine the individual models:

\[ \mathcal{T} = \pi_{\epsilon}(\min(\det(\tilde{L} \circ G))) \]
full model

combine the individual models:

\[ \mathcal{T} = \pi_\epsilon(\min(\det(\tilde{L} \circ G))) \]
chord generation example

melody->chord FST ($L$):
chord generation example

chord model FSA (G):
chord generation example

input melody sequence \( m = [e, d, c] \)

construct “linear chain” FST, \( M \), from melody:

\[
\begin{array}{c}
0 \quad e \quad 1 \quad d \quad 2 \quad c \quad 3
\end{array}
\]
chord generation example

compute $C = M \circ T$

\[
\begin{array}{c}
0 & \xrightarrow{e : \mathcal{E}} & 1 & \xrightarrow{d : \mathcal{E}} & 2 & \xrightarrow{e : C} & 5 \\
& & \xrightarrow{e : C} & 3 & \xrightarrow{d : G7} & 4 & \xrightarrow{c : C}
\end{array}
\]
chord generation example

compute $C = M \cdot T$
chord generation example

compute \( C = M \circ T \)
chord generation example

compute $C = M \circ T$
chord generation example

finding the accompaniment sequence from C:

- shortest path (most likely chords)
- randomly generate (uniform sampling or weights as negative log probabilities)
extension to rhythm

harmonic accompaniment model doesn’t incorporate rhythm, so we add an separate rhythmic model
extension to rhythm

use the same basic approach as in the case of generating harmonic accompaniment:

\[
\hat{r} = \arg \max_{r \in \Sigma^*} \Pr[r | s] = \arg \max_{r \in \Sigma^*} \Pr[s | r] \cdot \Pr[r]
\]

alphabet consists of quantized inter-onset times
The Harmonically Ecosystemic Machine

collaboration with Michael Musick

interactive installation that incorporates FST-based harmony and rhythm generation[2]

The Harmonically Ecosystemic Machine

Q - Responder

P - Listener

Melody

Chords

f - Reflector
The Harmonically Ecosystemic Machine

*Listener* and *Responder* modules implemented in SuperCollider

*Reflector/Decision Maker* module implemented in Python

uses OpenFst and OpenGrm libraries

communication via OSC
The Harmonically Ecosystemic Machine
The Harmonically Ecosystemic Machine

training data:

harmony: Rock Corpus dataset
rhythm: works by Mozart
The Harmonically Ecosystemic Machine

why?

aesthetics: create familiar harmony and rhythms for participants

practicality: limited number of useful datasets
The Harmonically Ecosystemic Machine installation is located in the UNT Music Building room 2009
adapting to real-time

computational efficiency more important in real-time than offline system

improving efficiency involves tradeoffs
adapting to real-time

reduce size of $L$ FST to improve efficiency:

key normalize melody/harmony

... but this requires key estimation, which is error-prone
adapting to real-time

reduce size of G FST to improve efficiency:

limit n-gram order

... but longer context may be preferable
adapting to real-time

length of melodic input also impacts efficiency

longer input melody is more computationally expensive

... but longer context generally provides better harmonizations

also impacts system latency
adapting to real-time

for our implementation:

key-normalized $L$ FST

key independent chord representation

n-gram order = 3

input melody length of 8 beats
conclusions

harmonic generation system was only offline

project was an opportunity to explore system in a real-time setting

practical and aesthetic benefits
Thanks!

Music Building room 2009

http://steinhardt.nyu.edu/marl/research/sonic_spaces

jpf211@nyu.edu