Center for Research in Entertainment and Learning

Discrete Musical Systems

Shlomo Dubnov

UCSD

CELFI Seminar, Lecture 2

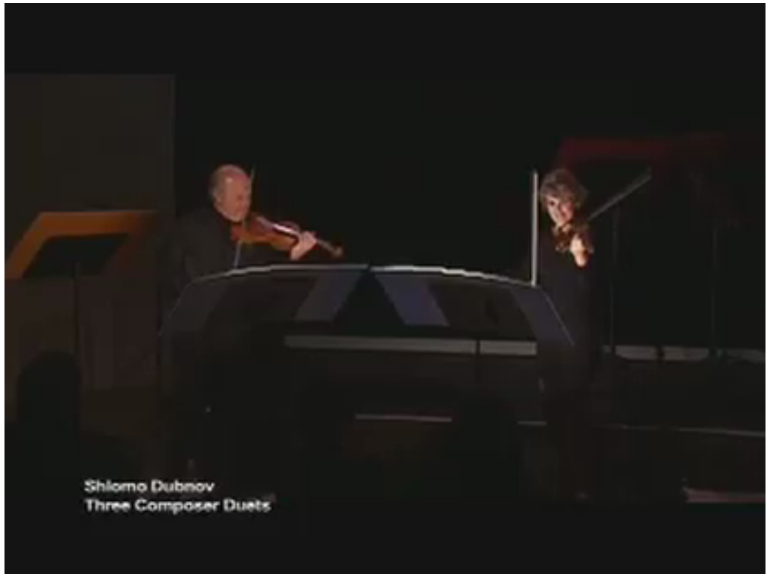
Style Modeling and Machine Improvisation

Machine learning of Variable Memory Markov models with focus on generative and interactive applications

Machine Improvisation

PyOracle

Memex: Composer Duos



OMax

ircam
Centre
Pompidou

Existing Systems

- Mimi
- Continuator
- Omax
- PyOracle
- ImproteK

First paper on Machine Improvisation

Guessing the Composer's Mind: Applying Universal Prediction to Musical Style

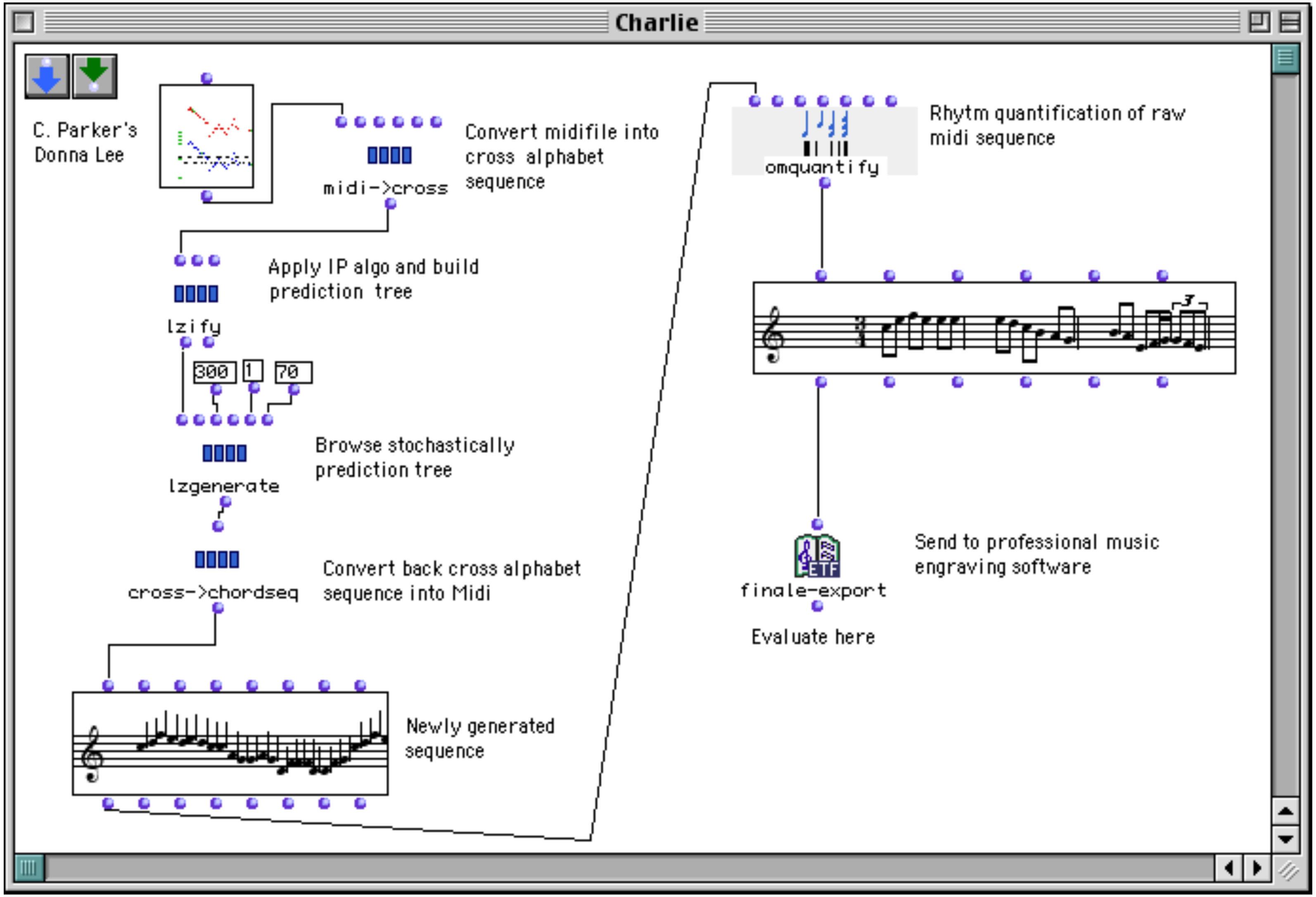
Gérard Assayag (Ircam) , Shlomo Dubnov (Ben Gurion Univ.), Olivier Delerue (Ircam)

Abstract

In this paper, we present a dictionary based universal prediction algorithm that provides a very general and flexible approach to machine learning in the domain of musical style. Such operations as improvisation or assistance to composition can be realized on the resulting representations.

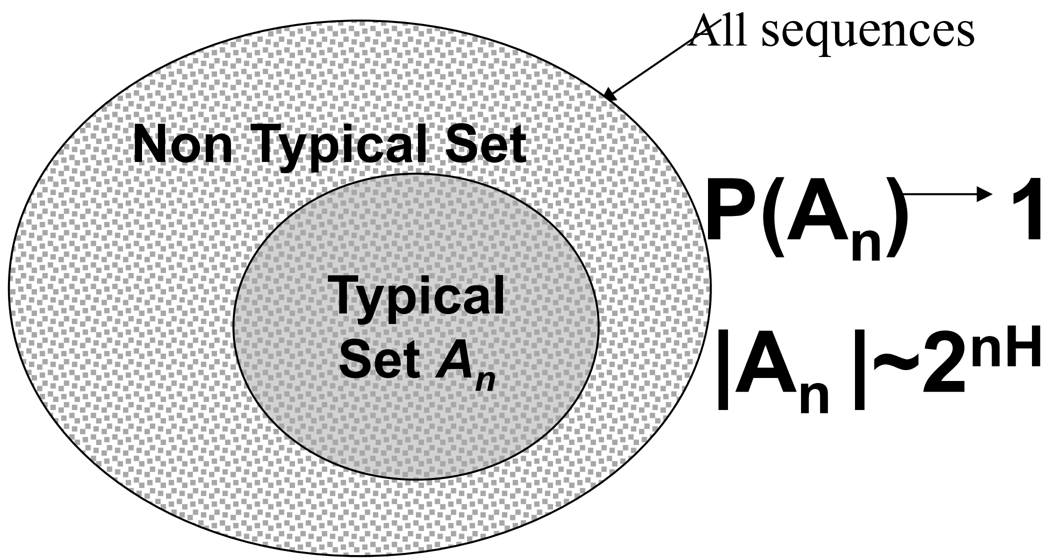
ICMC 1999

lzify


Charlie
C. Parker's
Donna Lee
Convert midifile into
cross alphabet
sequence
midi->cross
Apply IP algo and build
prediction tree
lzify
300
1
70
Browse stochastically
prediction tree
lzgenerate
Convert back cross alphabet
sequence into Midi
cross->chordseq
Newly generated
sequence
Rhytm quantification of raw
midi sequence
omquantify
Send to professional music
engraving software
finale-export
Evaluate here

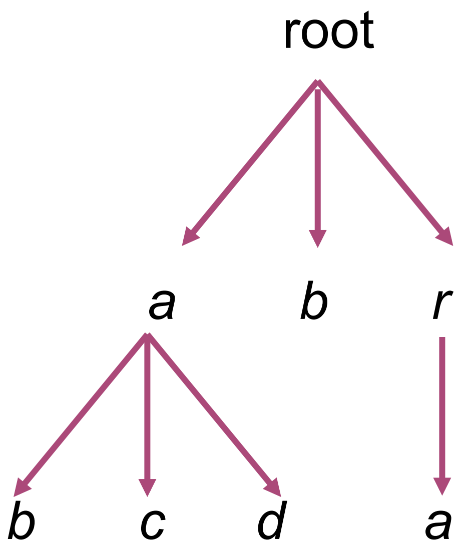
Style Learning Algorithms

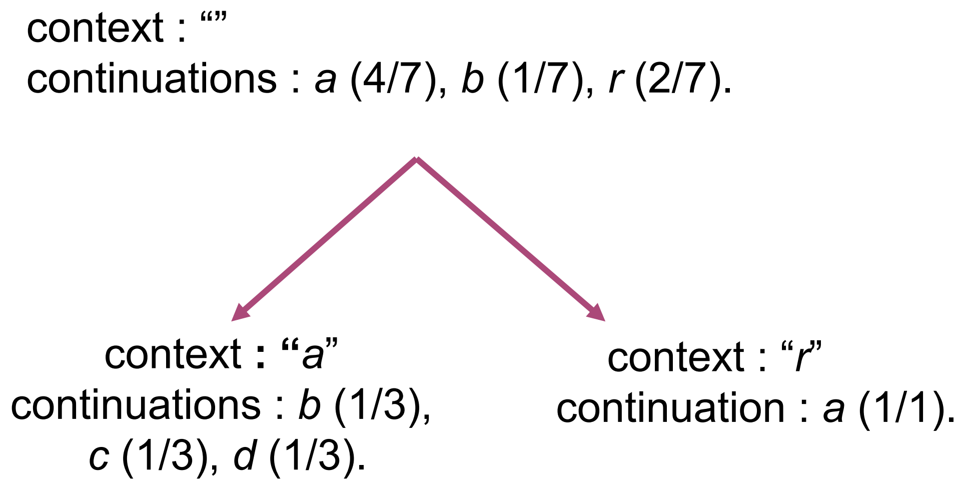
- Create Musical Generators from Examples that maintain *similarity* with the *style* of the original example



Suffix Automata using LZ

- Analysis of "*abracadabra*".





$$P(\text{generate "}abrac\text{"}) = P(a|\text{""})P(b|\underline{\mathbf{a}})P(r|ab)P(a|ab\underline{\mathbf{r}})P(c|abr\underline{\mathbf{a}}) = 4/7 \cdot 1/3 \cdot 2/7 \cdot 1 \cdot 1/3.$$

IPMotif

```
def IPMotif(text):
    """Compute an associative dictionary (the motif dictionary)."""

    dictionary = {}
    motif = ""
    result = []
    for c in text:
        motif = motif + c
        if motif in dictionary:
            # Increase count for existing motif
#            print '%s in dictionary' % motif
            dictionary[motif] += 1
        else:
            # Add motif to the dictionary.
            dictionary[motif] = 1
            motif = ""
#            print 'add %s to dictionary' % motif

    return dictionary
```

{'a': 4, 'ac': 1, 'b': 1, 'ad': 1, 'r': 2, 'ra': 1, 'ab': 1}

IPContinuation

```
def IPContinuation(dict1):
    """Compute continuation dictionary from a motif dictionary"""

    dict2 = {}
    for Wk in dict1:
        counter = dict1[Wk]
        W = Wk[:-1]
        k = Wk[-1]
        if W in dict2:
            dict2[W].append((k,counter))
        else:
            dict2[W] = [(k,counter)]
    dict2 = Normalize(dict2)
    return dict2


def Normalize(dict2):
    """Turns the counters in every element of dict2 to probabilities

    for W in dict2:
        cnt = [tup[1] for tup in dict2[W]]
        ttl = sum(cnt)
        for k,tup in enumerate(dict2[W]):
            dict2[W][k] = (tup[0],float(tup[1])/ttl)
    return dict2
```

{'': [('a', 0.57), ('b', 0.14), ('r', 0.28)], 'a': [('c', 0.33), ('d', 0.33), ('b', 0.33)], 'r': [('a', 1.0)]}

Markov

```
def Markov(text,order=0):
    """Compute a Markov models (fixed length motif dictionary)."""

    dict3 = {}
    for i in range(len(text)-order):
        W = text[i:i+order]
        k  = text[i+order]
        if W in dict3:
            if k in list(zip(*dict3[W])[0]):
                dict3[W][k] += 1
            else:
                dict3[W][k] = 1
        else:
            dict3[W] = {k:1}

    for x in dict3:
        dict3[x] = dict3[x].items()
    dict3 = Normalize(dict3)
    return dict3
```

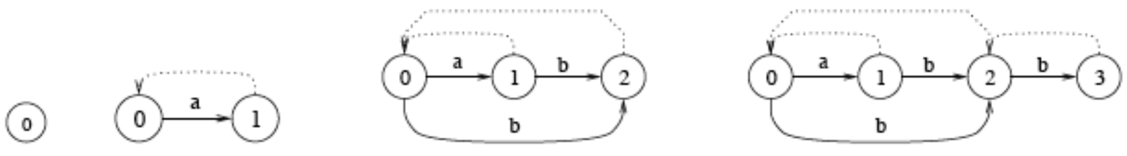
{'a': [('c', 0.25), ('b', 0.5), ('d', 0.25)], 'r': [('a', 1.0)], 'b': [('r', 1.0)], 'c': [('a', 1.0)], 'd': [('a', 1.0)]}

Factor Oracle

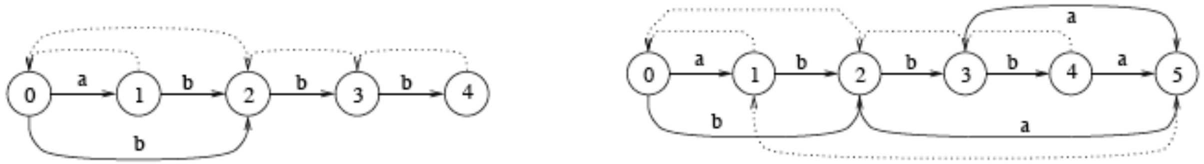
- Extension of a Suffix tree
- Automaton that accepts all factors (substrings) of a string
- Effective language is larger then the set of factors and is hard to characterize
- Auxiliary construction points to suffixes for each point along the sequence
- We use the suffixes for improvisation

Factor Oracle

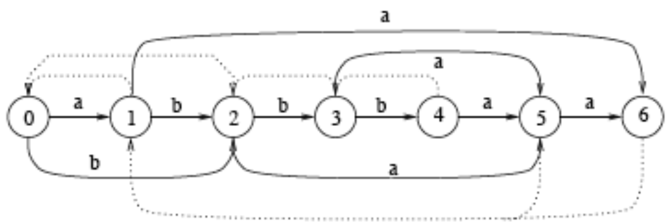
s = “abbbaab”



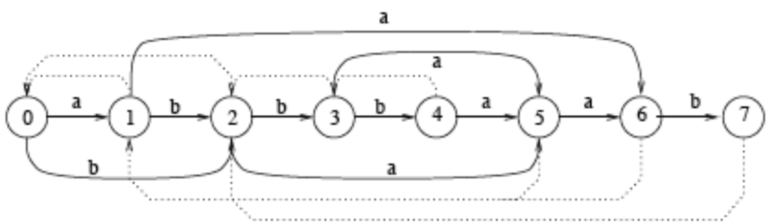
(a) (b) Add a (c) Add b (d) Add b



(e) Add b (f) Add a



(g) Add a

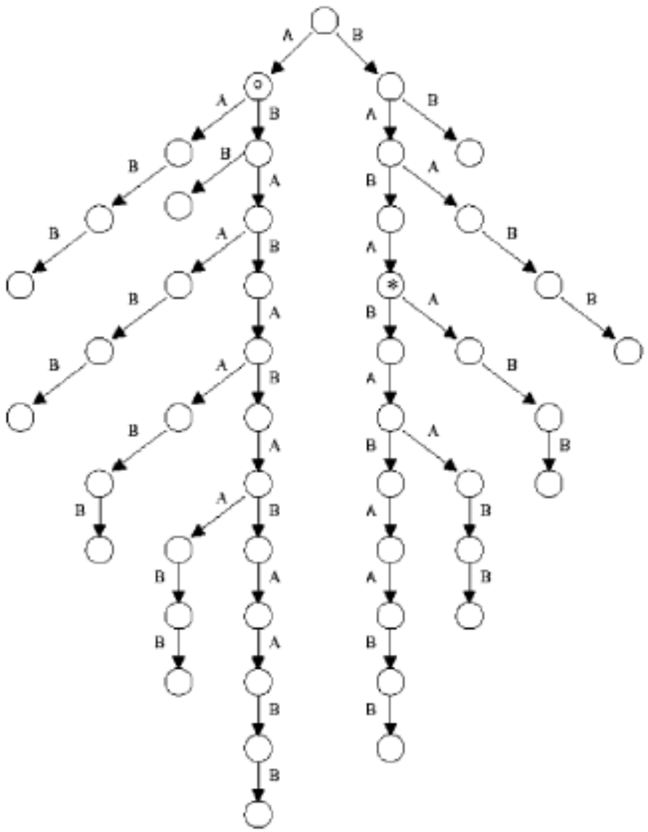


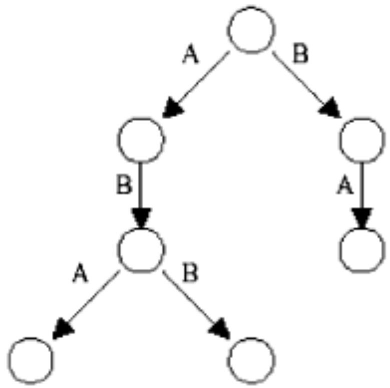
(h) Add b

```
[trn,sfx] = FO(s,a)

% Factor Oracle for sequence s
% input:
% s - string of numbers in range [1,a]
% a - size of the alphabet
% output:
% trm - transition matrix (forward)
% sfx - suffix vector (backward)


[s, kend, ktrace] = FOgen(trn,sfx,n,p,k)
% Generate new sequence using a Factor Oracle
% input:
% trn - transition table
% sfx - suffix vector
% n - length of new string
% p - probability of change
% k - starting point
% output:
% s - new sequence
% kend - end point
```

Suffix Tree versus IP

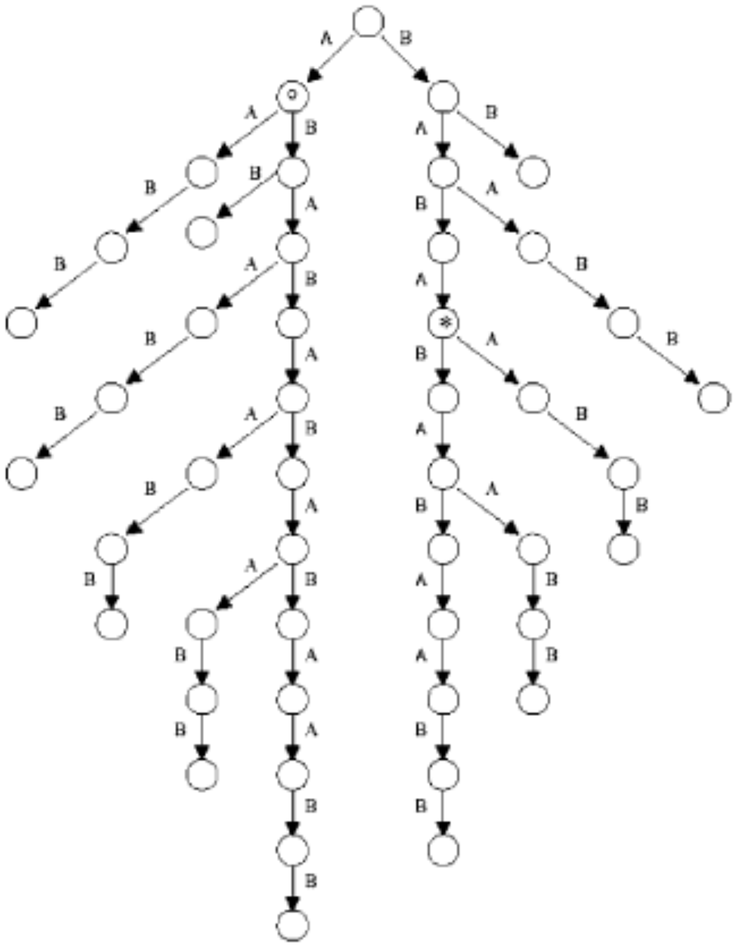
W = ABABABABAABB

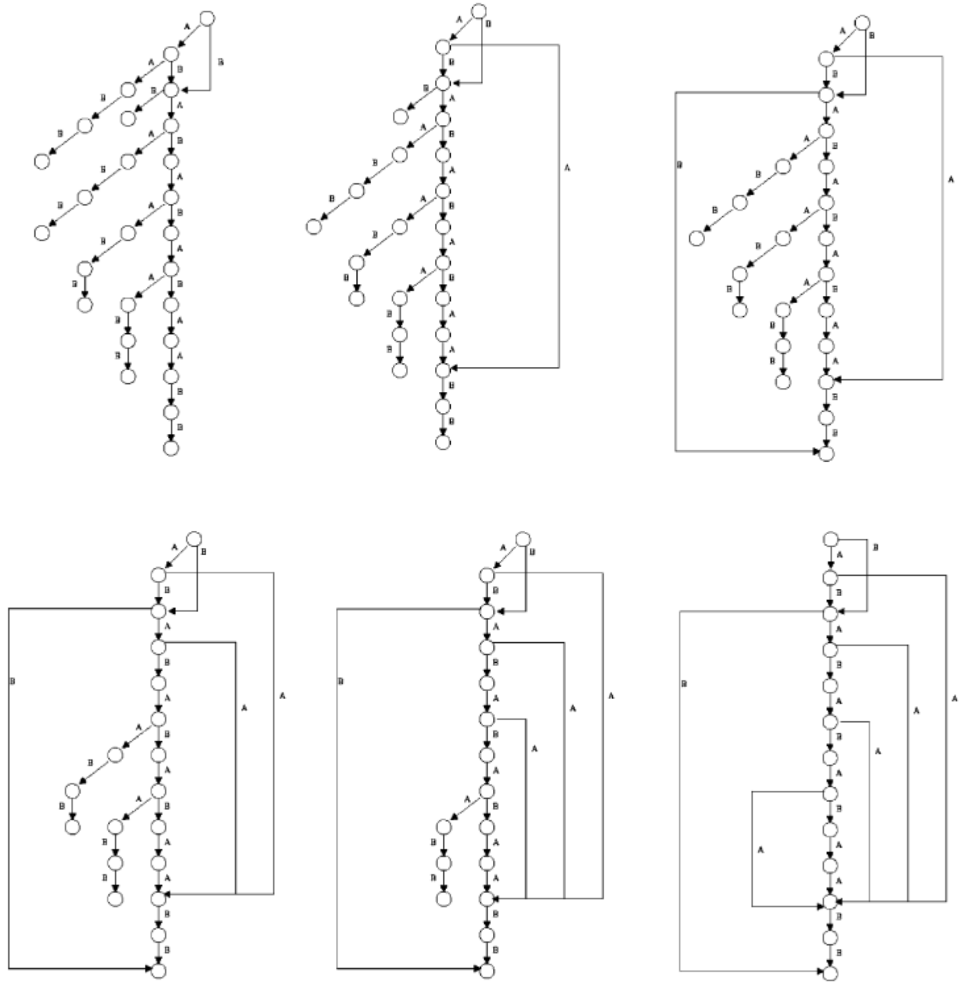




From Suffix Tree to Factor Oracle

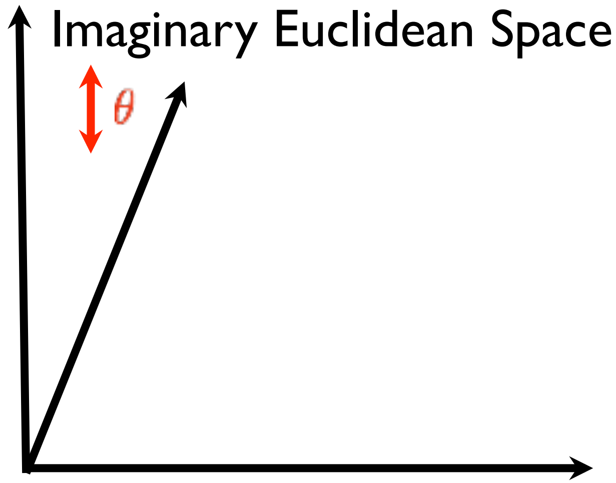
ababababaabb





FO versus Audio Oracle Learning

REALTIME Scheduler
Time = 0
No event!

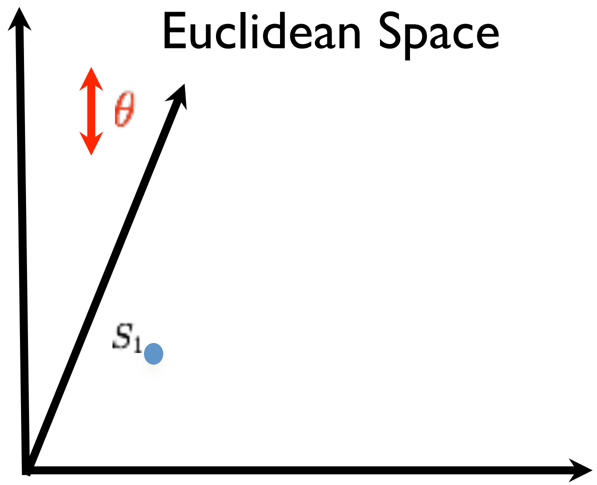


Sequential AO Learning:

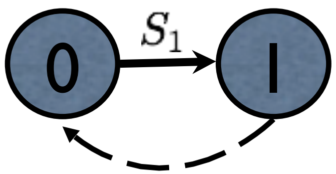
FO versus Audio Oracle Learning

Create suffix link 1

REALTIME Scheduler
Time = 1
Arrival of S_1



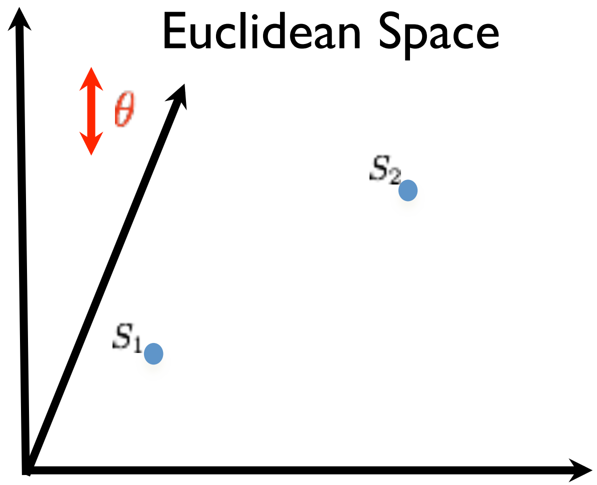
Sequential AO Learning:



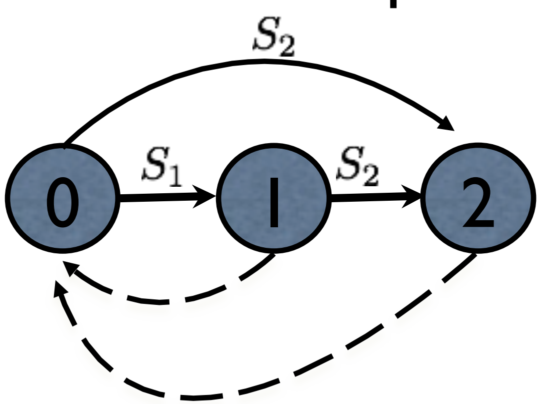
FO versus Audio Oracle Learning

Follow suffix 1
Create suffix 2
Create forward link 2

REALTIME Scheduler
Time = 2
Arrival of S_2



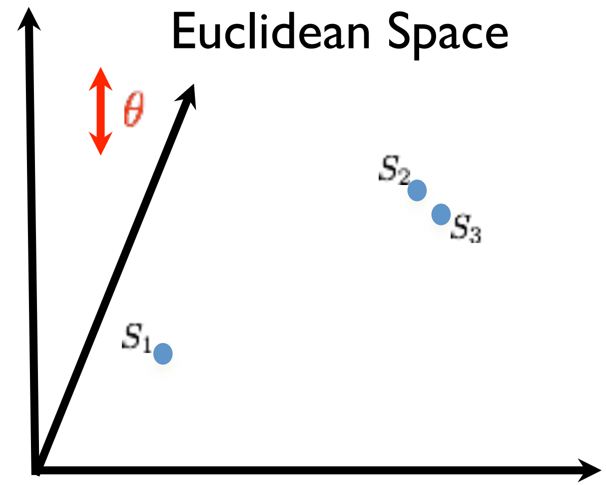
Sequential AO Learning:



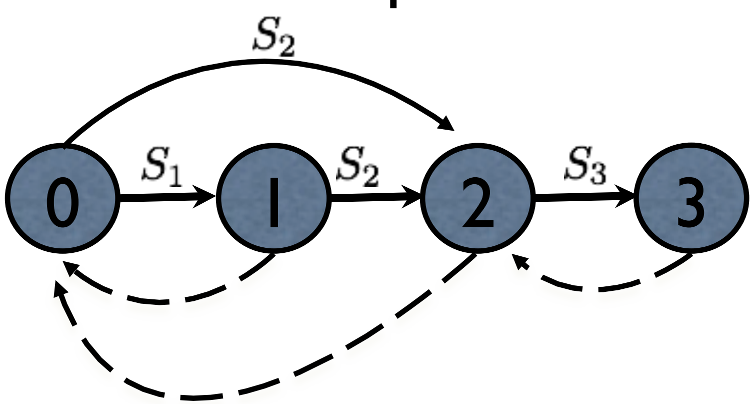
FO versus Audio Oracle Learning

Follow suffix 2
Follow forward link 2
Create suffix 3

REALTIME Scheduler
Time = 3
Arrival of S_3



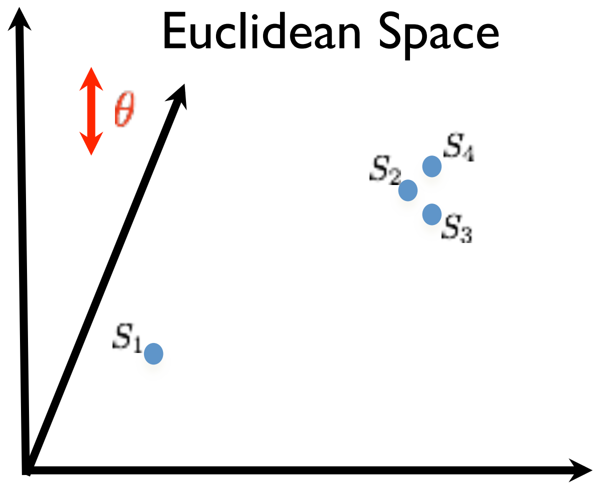
Sequential AO Learning:



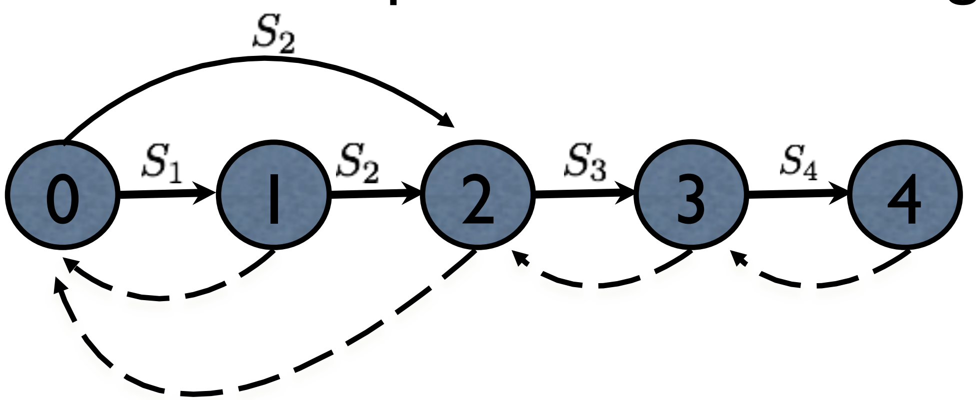
FO versus Audio Oracle Learning

Follow suffix 3
Follow forward link 3
Create suffix 4

REALTIME Scheduler
Time = 4
Arrival of S_4

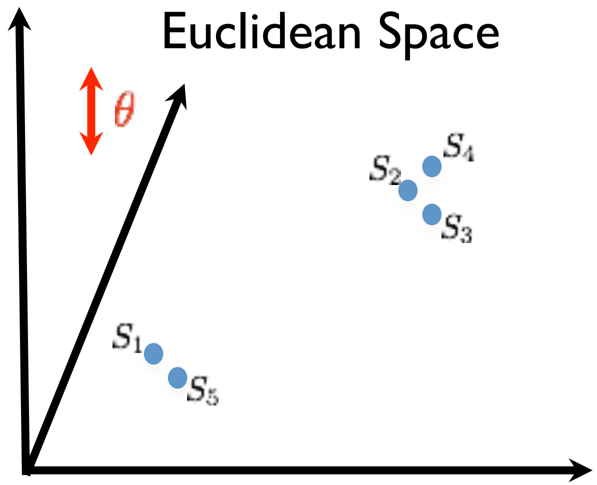


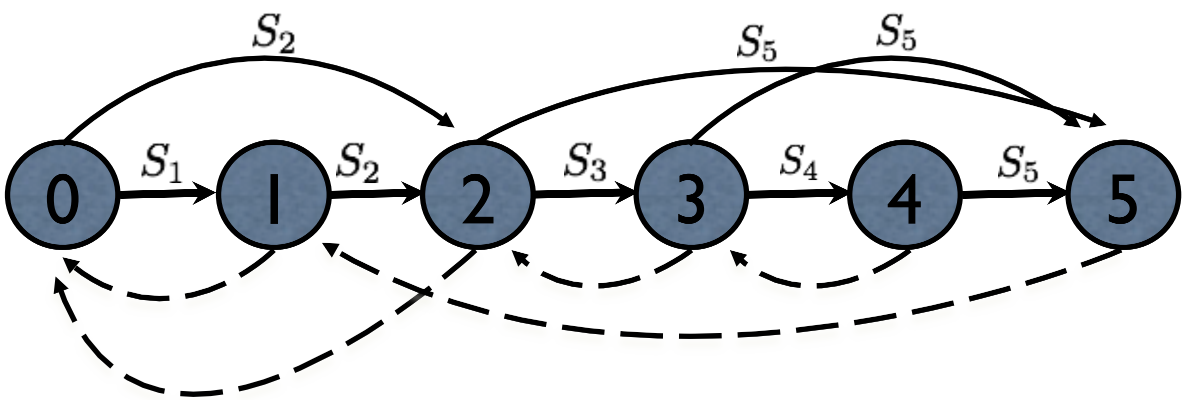
Sequential AO Learning:



FO versus Audio Oracle Learning

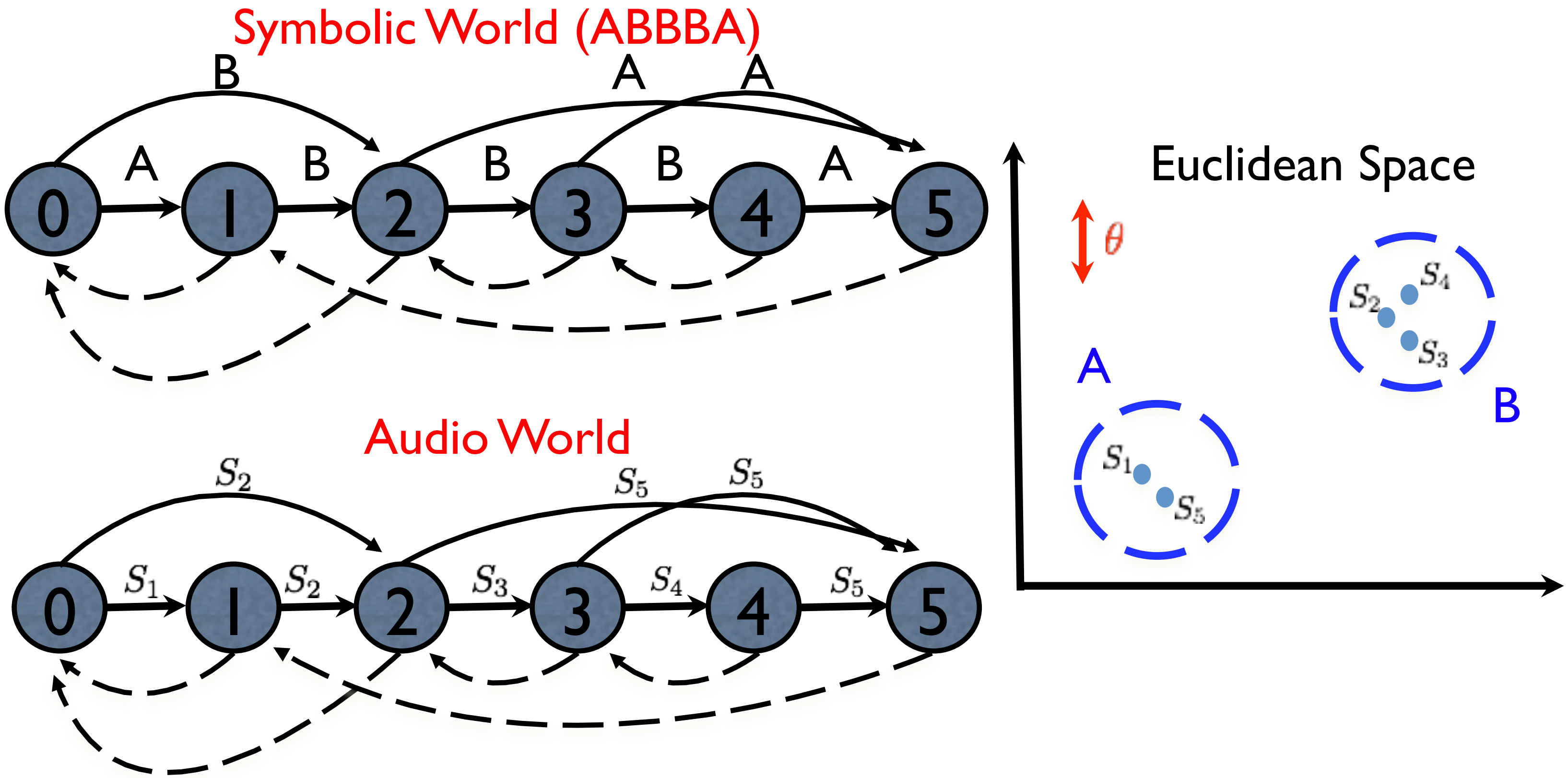
Follow suffix 4
Create forward link 5
Follow suffix 3
Create forward link 5
Follow suffix 2
Follow forward link 1
Create suffix 5



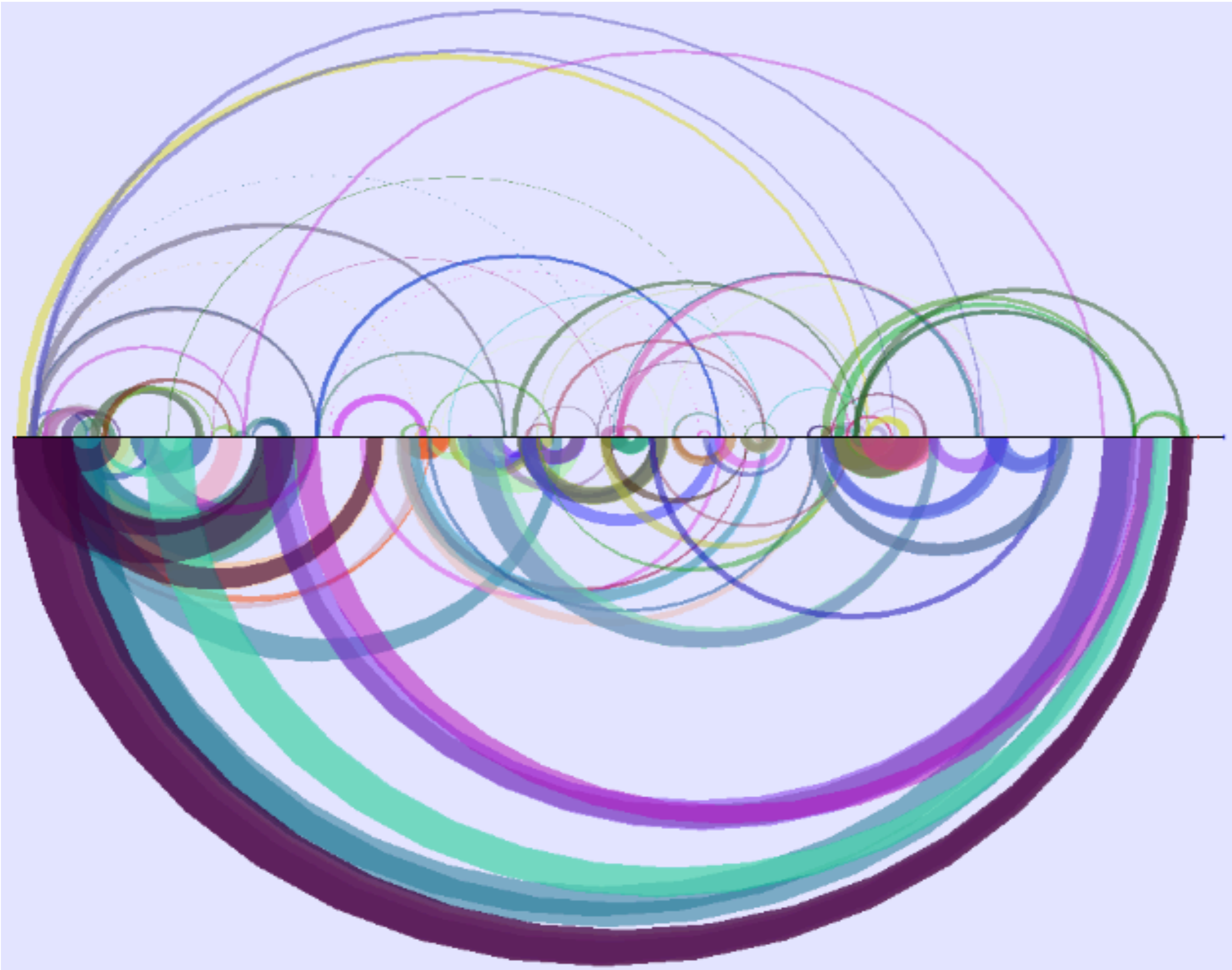


Implicit Quantization

AO requires a threshold for symbolization / quantization

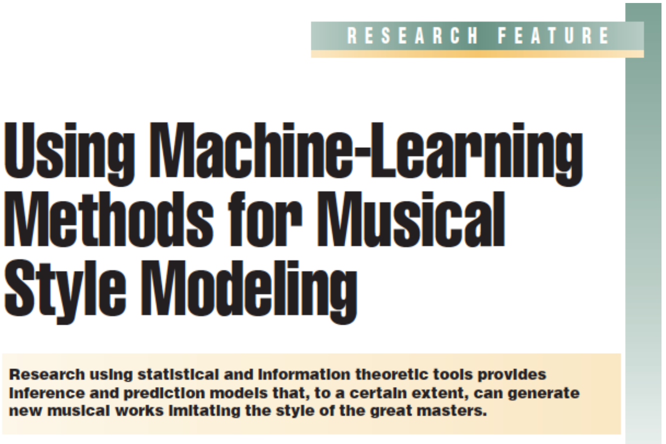


Memory Trails



INTERNATIONAL COMPUTER MUSIC ASSOCIATION

Guessing the Composer's Mind: Applying Universal Prediction to Musical Style
Assayag, Gérard; Dubnov, Shlomo; Delerue, Olivier, ICMC 1999


RESEARCH FEATURE

Using Machine-Learning Methods for Musical Style Modeling

Research using statistical and information theoretic tools provides inference and prediction models that, to a certain extent, can generate new musical works imitating the style of the great masters.

Shlomo Dubnov, Gerard Assayag, Olivier Lartillot, Gill Bejerano, IEEE Computer, 2003, Issue 10

Using Factor Oracle for Machine Improvisation, G. Assayag and S. Dubnov, Soft Computing 8 (9), 2004

Summary

- In LZify we are traversing the LZ tree using the longest suffix
- FO algorithm creates a “memory trail” in the form of suffix links pointing to longest repeated suffix (LRS)
- Suffix links and reverse suffix links constitute all points in a sequence that share a common history
- We can use these suffixes to “remix” the signal, i.e. “improvise”
- If we know how to measure similarity between signals, it is possible to generalize FO to Audio Oracle (FO over a metric space).

Challenges

- What if the radius of the balls is unknown?
  - Using FO on an Audio Signal requires symbolization / quantization prior to FO
- Audio Signal Can be analyzed in terms of multiple features.
  - Which feature to choose?
- Specifying User Interaction and Musical Form

Next: Information Dynamics criteria for symbolization using FO model