Week 5 – Music Generation and Algorithmic Composition

Roger B. Dannenberg
Professor of Computer Science and Art
Carnegie Mellon University

Overview

- Short Review of Probability Theory
- Markov Models
- Grammars
- Patterns
- Template-Based Music
- Suffix Trees
- Data Compression and Music Generation

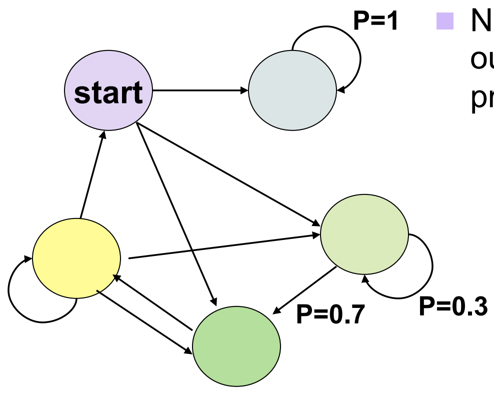
Probability

- Automatic Music Generation/Composition often uses probabilities
- Usual question: what's the most likely thing to do?
- P(x) is the "probability of x"
- P(x|y) is the "probability of x given y"
- Example: given the previous pitch in a melody, what is the probability of the next one?

Markov Chains

- One of the most basic sequence models
- Markov Chain has:
  - Finite set of states
  - A designated start state
  - Transitions between states
  - Probability function for transitions
- Probability of the next state depends only upon the current state (1st-order Markov Chain)
- Can be extended to higher orders by considering previous N states in the next state probability.

Markov Chain as a Graph



- Note that the sum of the outgoing transition probabilities is 1.

5

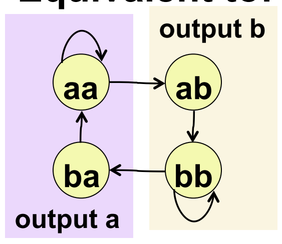
Spring 2019

Nth-Order Markov Chain

- Next state depends on previous N states, but you can always build an equivalent 1st-order Markov Chain with m^n states.

- P(a|aa) = 0.5, P(b|aa) = 0.5
- P(b|ab) = 1
- P(a|ba) = 1
- P(a|bb) = 0.5, P(b|bb) = 0.5

Equivalent to:



6

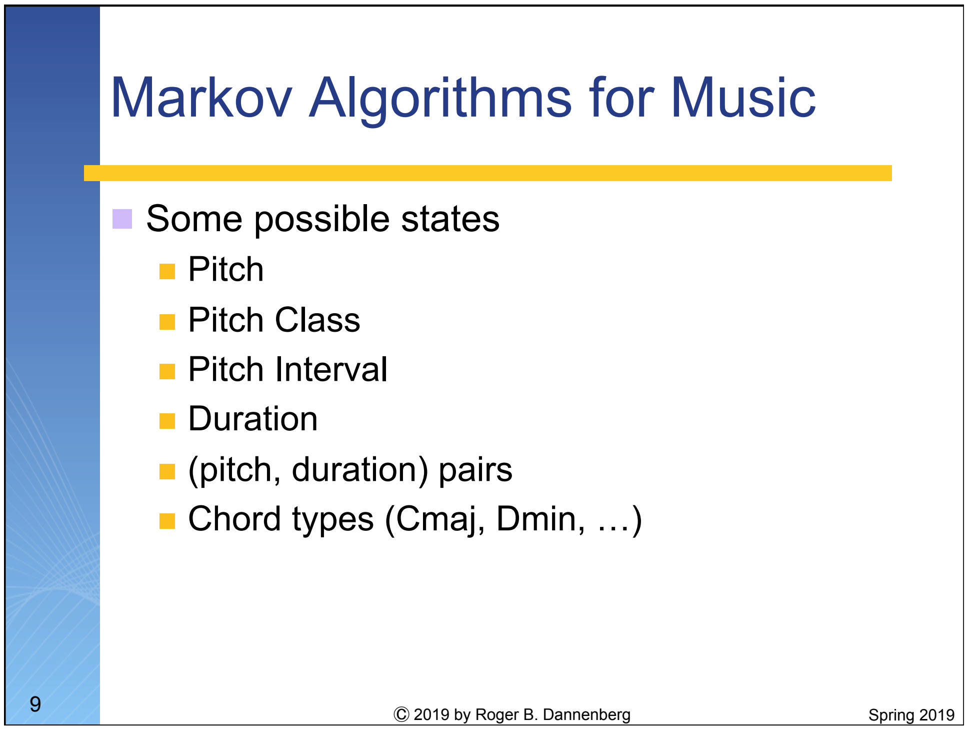
Spring 2019

Estimating Probabilities

- If a process obeys the Markov properties (or even if it doesn't), you can easily estimate transition probabilities from sample data.
- The more data, the better (law of large numbers)
- Let
  - n_A = no. of transitions observed from state A
  - n_{AB} = transitions from state A to state B
- Then
  - P(B|A) = estimated probability of a transition from state A to state B = n_{AB}/ n_A

The Last Note Problem

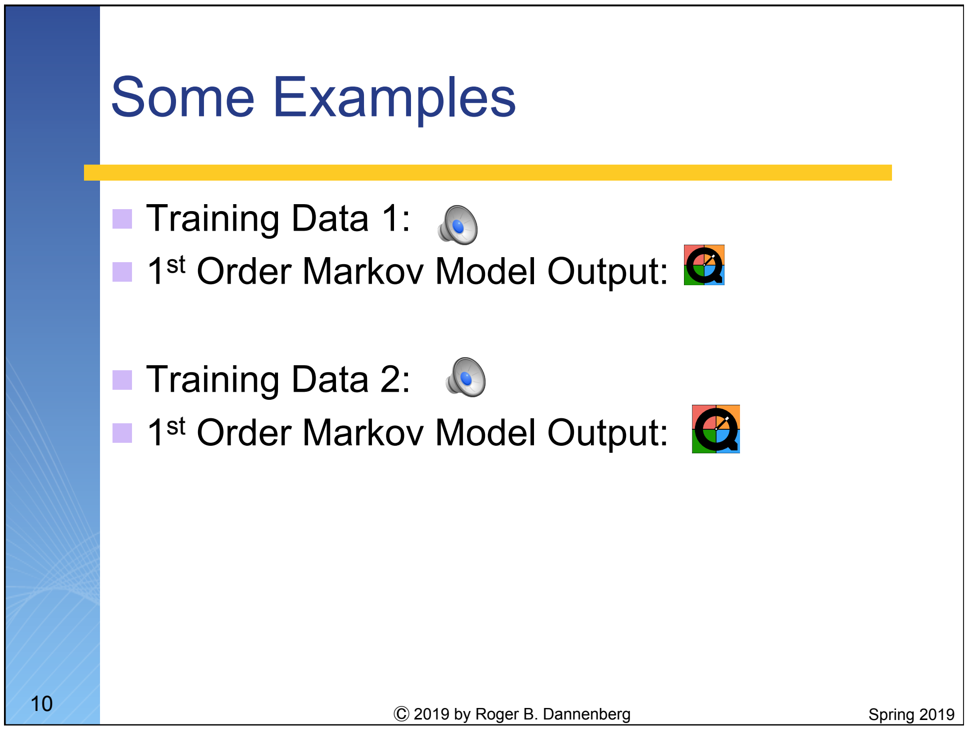
- Observations are always finite sequences
- There must always be a "last" state
- The last state may have no successor states ($n_{last\_state} = 0$)
- So P(B|A) = 0/0 = ?
- Solutions:
  - Initialize all counts to 1 (in the absence of any observation, all transition probabilities are equal), OR…
  - If there are no Nth-order counts, use (N-1)th-order counts, e.g. estimate $P(B|A) \approx P(B) = n_B/n$, where n is total number of observations, OR…
  - Pretend the successor state of the last state is the first state -- now every state leads to at least one other.



Markov Algorithms for Music

- Some possible states
  - Pitch
  - Pitch Class
  - Pitch Interval
  - Duration
  - (pitch, duration) pairs
  - Chord types (Cmaj, Dmin, …)

9 © 2019 by Roger B. Dannenberg Spring 2019



Some Examples

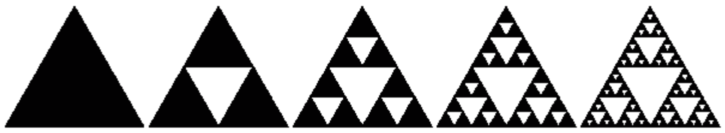
- Training Data 1:
- 1st Order Markov Model Output:

- Training Data 2:
- 1st Order Markov Model Output:

10 © 2019 by Roger B. Dannenberg Spring 2019

Mathematical Systems

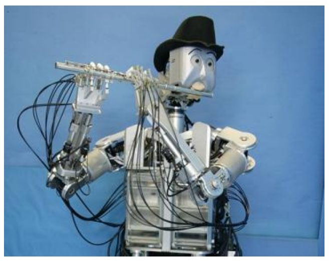
- Sierpinski's Triangle



- Music: start with one note. Divide into 3 parts, divide each part into 3 parts, …. On each division into 3 parts, transpose the pitch by 3 different values. Keep the original pitch as well, so we have one long note and 3 short ones (each of which has 3 shorter notes, etc.)

Mapping Natural Phenomena to Music

- Example: map image pixels to music
  - Sudden change in "red" -> start a note
  - Pitch comes from "blue"
  - Loudness comes from "green"
  - Repetitive structure because adjacent scan lines are similar



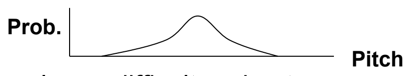
Chromatic

Diatonic

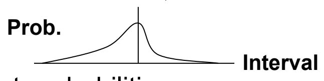
Microtonal

David Temperley's Probabilistic Melody Model

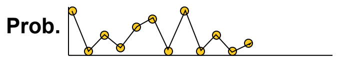
- There are several probability distributions that might govern melodic construction:
  - The voice has limited range: central pitches are more likely:



  - Large intervals are difficult and not so common, so we have an interval distribution:



  - Different scale steps have different probabilities:



- We can combine these probabilities by multiplication to get relative probabilities of the next note
- Distributions can be estimated from data.

Grammars for Music Generation

- Reference: Curtis Roads, *The Computer Music Tutorial*
- Formal Grammar Review
  - Set of tokens
  - The null token Ø
  - Vocabulary V = tokens U Ø
  - Token is either terminal or non-terminal
  - Root token
  - Rewrite Rules: $\alpha \rightarrow \beta$

Grammars (2)

- Context-Free Grammars
  - Left side of rule is a single non-terminal
- Context-Sensitive Grammars
  - Left side of rule can be a string of tokens, e.g.
    $A\alpha A \rightarrow A\rho B$
    $B\alpha C \rightarrow B\sigma C$
- Grammars can be augmented with procedures to express special cases, additional language knowledge, etc.

15

Spring 2019

Music and Parallelism

- Conventional (formal) grammars produce 1-dim strings
- Replacement is always in 1-dim ($a \rightarrow b\ c$)
- Multidimensional grammars are simple extension:
- $a \rightarrow b,c$ —*sequential combination*
- $a \rightarrow b|c$ —*parallel combination*

16

Spring 2019

Non-Local Constraints

- This is a real limitation of grammars, e.g.
  - Making two voices (bass and treble) have same duration
  - Making a call and response have same duration
  - Expressing an upward gesture followed by a downward gesture
- Procedural transformations and constraints on selection are sometimes used

17

Probabilistic Temporal Graph Grammars (D. Quick & P. Hudak)

- Local constraints added with new type of rule:
  let *x* = *A* **in** *xBx*
  is not the same as ABA because *x* is expanded once and used twice, whereas in ABA, each A can be expanded independently.
- Durations are handled with superscripts, e.g.
  $I^t \rightarrow I^{t/2}\ V^{t/2}$
  means that non-terminal I with duration *t* can be expanded to I V, each with duration *t*/2.

18

Example

- S^d -> S^d P^d | P^d
- P^d -> let x = Q^d in x x
- Q^d -> $Q^{d/2}$ $Q^{d/2}$ | B^d | R^d
- where B is a beat, R is a rest
- A problem(?): Average max depth is ~8, but sensible limit might be ~5 (thirty-second notes)
- With 1/32 lower bound for d:

```
((R 0.125)         (B 0.0625)         (B 0.0625)
 (B 0.015625)      (B 0.03125)        (B 0.03125)
 (B 0.0078125)     (B 0.0078125)      (R 0.03125)
 (B 0.00390625)    (R 0.0078125)      (B 0.0625)
 (R 0.000976562)   (B 0.00390625)     (R 0.0625)
 (B 0.000976562)   (R 0.00195312)     (B 0.25)
 (R 0.00195312)    (R 0.00195312)     (B 0.5)
 (B 0.00390625)    (B 0.0078125)      (B 0.0625)
 (B 0.00390625)    (B 0.03125)        (B 0.03125)
 (R 0.00390625)    (R 0.03125)        (R 0.03125)
 (R 0.00195312)    (R 0.0625)         (B 0.0625)
 (B 0.00195312)    (B 0.0625)         (R 0.0625)
 (B 0.015625)      (R 0.25)           (B 0.25)
 (B 0.0625)        (R 0.25)           (R 1)
 (B 0.03125)       (R 1)              (R 1))
 (B 0.0078125)     (R 1)
 (R 0.0078125)     (B 0.25)
 (B 0.00390625)    (R 0.25)
 (R 0.00195312)    (B 0.0078125)
 (R 0.00195312)    (B 0.00390625)
 (B 0.0078125)     (R 0.00390625)
 (B 0.03125)       (B 0.015625)
 (R 0.03125)       (B 0.03125)
 (R 0.0625)        (B 0.0625)
 (B 0.0625)        (B 0.0625)
 (R 0.25)          (R 0.0625)
 (R 0.25)          (R 0.25)
 (R 0.125)         (B 0.25)
 (B 0.015625)      (R 0.25)
 (B 0.0078125)     (B 0.0078125)
 (B 0.00390625)    (B 0.00390625)
 (R 0.000976562)   (R 0.00390625)
 (B 0.000976562)   (B 0.015625)
 (R 0.00195312)    (B 0.03125)
 (B 0.00390625)    (B 0.0625)
 (B 0.00390625)    (B 0.0625)
 (R 0.00390625)    (R 0.0625)
 (R 0.00195312)    (R 0.25)
 (B 0.00195312)    (B 0.5)
 (B 0.015625)
```

Implementation of Grammars

- Remember, we're talking about *generative* grammars
- Maybe you learned about *parsing* languages described by a formal grammar
- Generation is simpler than parsing
- Simplest way is by coding grammar rules as subroutines

Implementation Example

- A → A B
- A → B
- B → a
- B → b

```
def A():
    if random() < pAB
        A()
        B()
    else
        B()
def B()
    if random() < pa
        output("a")
    else
        output("b")
```

Assessment

- "Rewrite rules and the notion of context-sensitivity are usually based on hierarchical syntactic categories, whereas in music there are innumerable nonhierarchical ways of parsing music that are difficult to represent as part of a grammar." (Roads, 1996)

Pattern Generators

- Flexible way to generate musical data
- No formal learning, training, or modeling procedure
- Most extensive implementations are probably Common Music, a Common Lisp-based music composition environment, and Nyquist (familiar from my Intro to Computer Music class)

23

Cycle

- Input list: (A B C)
- Rule: repeat items in sequence
- Output: A B C A B C …

- Example:

24

Random

- Input list: (A B C)
- Rule: select inputs at random with replacement
- Input items can have weights
- Output can have maximum/minimum repeat counts
- Output: B A C A A B C C …
- Example: Example (12-tone):

Palindrome

- Input list: (A B C)
- Rule: repeat items forwards and backwards
- Output: A B C B A B C B …
- Additional parameters tell whether to repeat first and last items.
- Example:

Heap

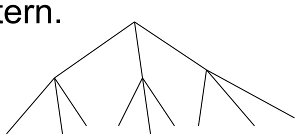
- Input list: (A B C)
- Rule: select items at random without replacement (until empty)
- Output: A B C, B C A, C B A, …
- Example:

Markov

- Input list: ((A -> B C) (B -> C) (C -> A B) …)
- Rule: generate a Markov chain
- Transitions may have weights
- Output: A B C A C B C B C A …

Nested Patterns

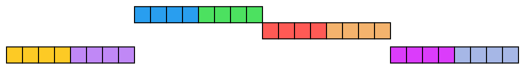
- Pattern items can be patterns, e.g.
  - Replace every element in a cycle pattern with a random pattern.

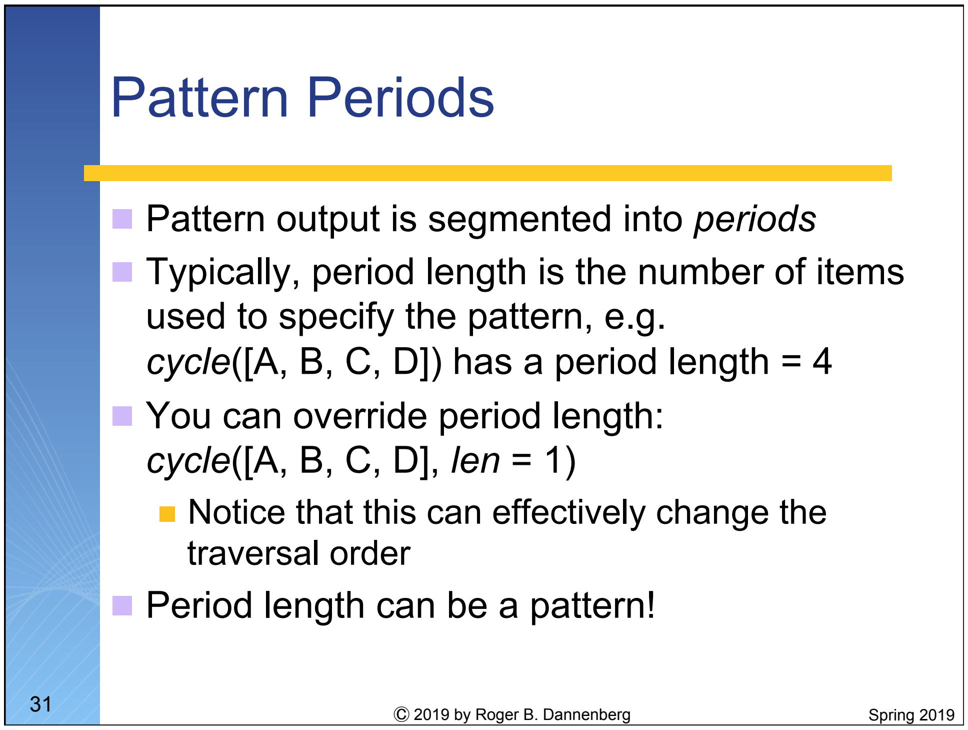


- What's the traversal order?
  - generate one *period* of items from sub-pattern before advancing to the next item in the pattern

Example

- Every set of 4 pitches is a permutation of {C, D, x, G}, where x is randomly selected from E, F, A, Bb
- Pick permutations of 4 and repeat them 4 times
- (now we have units of 16 pitches: 4 repetitions of 4 pitches)
- Every two units of 16 (i.e. every 32 notes), we apply the next transposition from the sequence 0, 5, 7, 0
- Here are 2 cycles of that (256 notes)
- A picture of 1 cycle of 128:

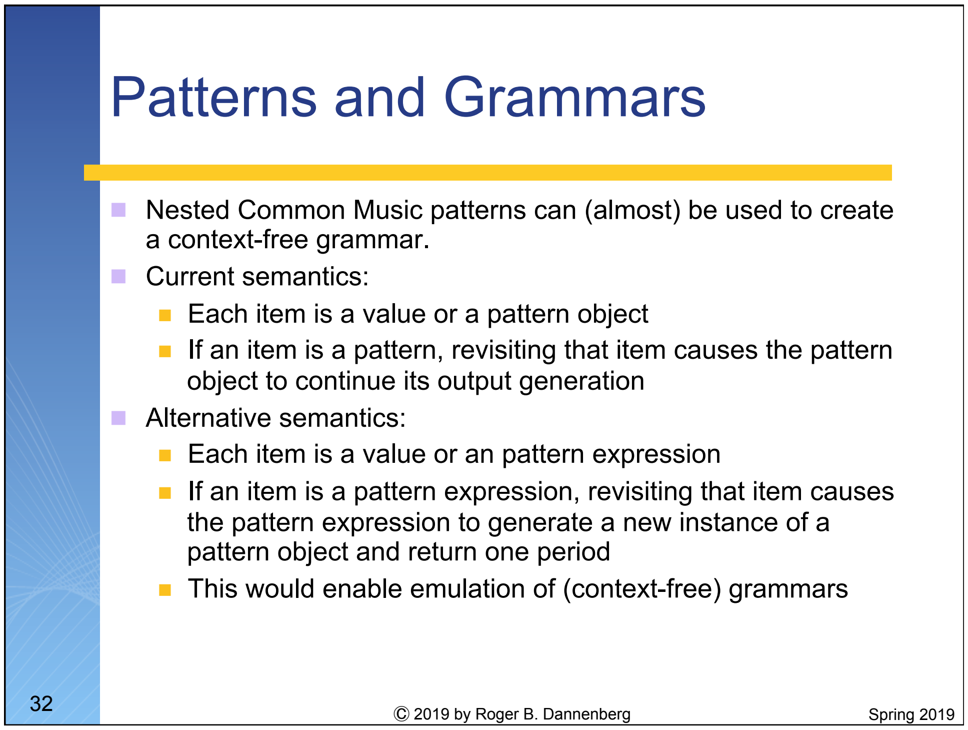




Pattern Periods

- Pattern output is segmented into *periods*
- Typically, period length is the number of items used to specify the pattern, e.g.
  cycle([A, B, C, D]) has a period length = 4
- You can override period length:
  cycle([A, B, C, D], *len* = 1)
  - Notice that this can effectively change the traversal order
- Period length can be a pattern!

31 Ⓒ 2019 by Roger B. Dannenberg Spring 2019



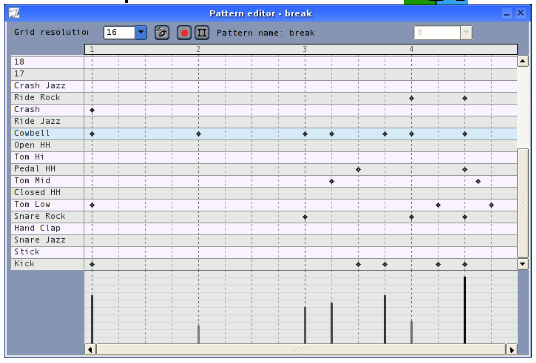
Patterns and Grammars

- Nested Common Music patterns can (almost) be used to create a context-free grammar.
- Current semantics:
  - Each item is a value or a pattern object
  - If an item is a pattern, revisiting that item causes the pattern object to continue its output generation
- Alternative semantics:
  - Each item is a value or an pattern expression
  - If an item is a pattern expression, revisiting that item causes the pattern expression to generate a new instance of a pattern object and return one period
  - This would enable emulation of (context-free) grammars

32 Ⓒ 2019 by Roger B. Dannenberg Spring 2019

Template-Based Music

- Music can be constrained by templates, grids, scales, harmony, etc.
- Example: drum machine


Pattern editor - break

Grid resolution 16 Pattern name: break

Roland CR-78 (1978)

33

Chord Templates

- Chords are just sets of pitches
- Can be described as set of pitch classes
  - If i is MIDI key number, PitchClass(i) = i mod 12

    C-major = {0, 4, 7} C-minor = {0, 3, 7}

    D-major = {2, 6, 9} D-minor = {2, 5, 9}

    E7-flat9 = {2, 4, 5, 8, 11}
- The bottom-most or *bass* note is important, so usually you also want to specify that too.

34

Bass Lines

- Chords often specify which note is the lowest (bass) note
- Common to use the root or 3rd of the chord
- Bass often “outlines” the chord
  - E.g. alternate root and fifth, or
  - Root, third, fifth, third pattern, etc.
- Use templates as in drums and chord patterns.
- Apply rules from harmony, counterpoint, jazz, rock, …

35 Ⓒ 2019 by Roger B. Dannenberg Spring 2019

Arpeggiators

- Cycle through chord tones
  - E.g. C-major = {0, 4, 7}, so play 0, 4, 7, 0, 4, 7
  - or 0, 4, 7, 4, 0, 4, 7, …
  - or 0, 4, 7, 12, 0, 4, 7, 12, …

Examples from Jim Aiken, secrets-of-the-arpeggiator.html

Example from http://www.ucapps.de/howto_sid_wavetables_3.html

36 Ⓒ 2019 by Roger B. Dannenberg Spring 2019

Melody

- Very prominent aspect of music, therefore difficult
- Chords imply scales:
  - Simple chords have 3 or 4 pitch classes (out of 12)
  - Scales are typically 7 pitch classes
    - do, re, mi, fa, so, la, ti, (do)
  - Constrain melody to scale
- Intervals are typically small -- stepwise motion
  - Can use histogram for interval selection
  - Or Markov chain for pitch sequence generation
- Rhythm is important too:
  - Markov Chain
  - Templates
  - Maybe 4-bar rhythm patterns from a database

Practical Algorithm Music Generation

- We've seen some interesting theory
- How does this all work in practice?
- Assume: goal is to generate "popular" music: rock, techno, jazz, dance, etc.
  - "experimental" music has fewer normative rules and more focus on new sounds, new structures, new concepts
  - "classical" music often includes development, transformations, themes and variation, which are very challenging
- Let's look at a direct rule- and probability-based method based on Friberg and Elowsson
- THIS IS NOT THE ONLY WAY!

Elowsson and Friberg, "Algorithmic Composition of Popular Music," 2012.

Algorithm Overview

- Make a structural plan: phrases, repetitions, similar rhythms
- Work phrase-by-phrase:
  - Compute rhythm track
  - Compute chords
  - Compute melody
- Add a little bit of search and evaluations
- Almost everything is a random weighted choice based on conditional probabilities

1. Overall Structure

- Currently, overall structure is simply selected
- A “little language” is used to express structure:
  - Same letter means high probability of the same melodic contour (same intervals)
  - A number means copy the rhythm and accents of the numbered phrase
- E.g. AB1CCAB means B mirrors rhythm of A (phrase 1), C repeats, and the final A and B mirrors the first A and B
- Duration of each phrase can be 2 or 4 measures.

2. Rhythmic Structure

- All measures in 4/4 time
- Represented as an array of 16th notes
- E.g. a 4 measure phrase is array(4 * 16)
- Kick (bass) drum every 2 beats,
- Pick some extra kick drum beats and add them

3. Chord Structure

- Only C major, D and E minor, F and G major, and A minor chords are generated
- Markov Model

```
CHORD_TRANSITION = [
#     C   Dm   Em    F    G   Am
  [ 24, 35,   0, 20, 70,   5 ], # to C
  [  2,  2,   5,  1,  1,   5 ], # to Dm
  [  2,  1,   0,  1,  2,   1 ], # to Em
  [ 39,  4, 85,  1, 13, 49 ], # to F
  [ 20, 86,   2, 76,  1, 39 ], # to G
  [ 35,  4,   8,  1, 14,   1 ]] # to Am
```

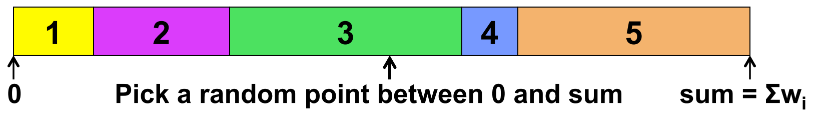
- Final chord is C major

4. Melodic Structure

- Compute both pitch and duration: computes probability for each combination of 15 pitches and 16 durations (1 to 16).
- For each of 15*16 pitch/duration combinations:
  - p = 1
  - For each ith melody rule:
    - p = p * P_i(pitch, duration)
- Then select according to computed probabilities

An Aside: Weighted Selection

- Given an array weights, choose an index, the likelihood of which is proportional to the weight
- The algorithm as a picture:



- In serpent:

```
require "prob”
print pr_weighted_choice([2, 3, 1.5, 0.1, …])
```

4b. Summary of Melody Rules 1

- Ambitus: discourage extremes of pitch range
- Harmonic: Is the note compatible with chord?

```
HARMONIZATION = [
// C     D     E     F     G     A     B
 [0.94, 0.30, 0.95, 0.16, 0.87, 0.26, 0.15],  // C
 [0.20, 0.90, 0.26, 0.86, 0.24, 0.88, 0.02],  // Dm
 [0.01, 0.18, 0.87, 0.09, 0.89, 0.24, 0.83],  // Em
 [0.90, 0.26, 0.18, 0.82, 0.29, 0.99, 0.01],  // F
 [0.28, 0.92, 0.28, 0.27, 0.95, 0.30, 0.75],  // G
 [0.92, 0.28, 0.85, 0.03, 0.25, 0.91, 0.20]]  // Am
```

- Interval:

```
INTERVAL_PROB = [0.2, 0.5, 0.3, 0.2, 0.15, 0.12, 0.03, 0.06]
```

4b. Summary of Melody Rules 2

- Interval Harmonic:
  - Prefer that larger intervals go up, prefer smaller going down
  - Avoid “unusual” intervals – e.g. 7th
  - Avoid intervals larger than 2nds with no chord tone
  - Larger intervals should be in the chord
- Duration:
  - Avoid 16th notes at fast tempo
  - Shorter durations favored over larger ones

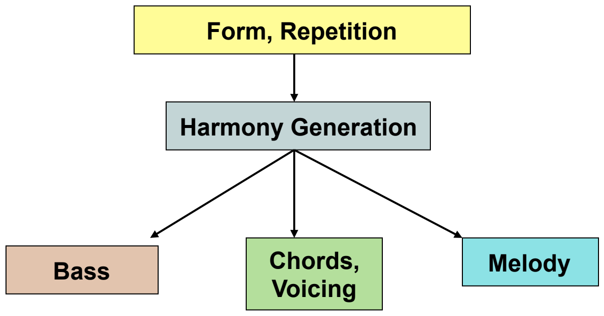
4b. Summary of Melody Rules 3

- Position/Duration: do not start and end on an odd 16th note beat position
- Harmonic Compliance/Duration:
  - Shorter notes favor dissonance
  - Longer notes favor consonance (with chord)
- Interval/Duration: larger intervals imply longer durations
- Phrase Arch: overall melodic contour (not implemented yet)

4b. Summary of Melody Rules 3

- Melodic Resolution: melody should approach final note with small intervals
- Resolve to Tonic: melody should end with C
- Metrical Salience: favor notes on strong metrical positions
- Mirror Intervals: if structure dictates a “mirror” phrase, e.g. “AABA”, all the “A”s should have similar interval sequences.

Constraints, Context, Form



Plan:
- **determine form from top down**
- **generate harmony for different sections**
- **fill in bass, chords, melody according to harmony**

Grammar-based Form Generation

- Plan:
  - Generate form from grammar
  - Control copies at different levels
- S = A1 A2 B A2
- A1 = C R1
- A2 = C R1'
- B = tr(B1, 9) tr(B1, 7) |
  tr(B2, x) tr(B2, x) tr(B2, y) tr(B2, 5)
- A1, A2 are 8 measures,
- B1 is 4 measures, B2 is 2 measures

Representation

- Time in beats
- Easy to append and merge
- [time-origin, duration, data-type, event-array]
  - data-type: 'chord', 'note'
  - Chord: [time-offset, duration, array-of-pcs]
  - Note: [time-offset, duration, pitch]
- Notes:

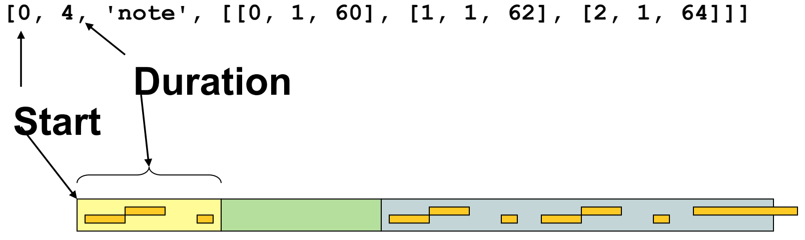
```
[0, 4, 'note', [[0, 1, 60], [1, 1, 62], [2, 1, 64]]]
```

- Chords:

```
[0, 4, 'chord', [[0, 2, [0, 4, 7]], [2, 2, [0, 3, 7]]]]
```

Representation

- `[0, 4, 'note', [[0, 1, 60], [1, 1, 62], [2, 1, 64]]]`



Explicit start/duration allows us to represent measures that are not completely full, silence, or even measures where notes extend into the next block

Manipulation: Code Example

```
// note: be sure to understand shallow vs deep copy
def sc_shift(s, shift)
    var events = []
    for e in sc_events(s)
        events.append([e[0] + shift, e[1], e[2]])
    return [s[0] + shift, s[1], s[2], events]

def sc_merge(a, b)
    sc_check_compatible(a, b)
    var start = min(sc_time(a), sc_time(b))
    var end = max(sc_end(a), sc_end(b))
    return [start, end - start, a[2],
            (sc_events(a) + sc_events(b)).sort()]

def sc_append(a, b)
    sc_merge(a, sc_shift(b, sc_end(a) - sc_time(b)))
```

Example: Generating Repeating Chord Progression

```
one = [0, 2, 'chord', [[0, 2, [0, 4, 7]]]]
two = [0, 2, [2, 5, 9]]
three = [0, 2, [4, 7, 11]]
four = [0, 2, [5, 9, 0]]
five = [0, 2, [7, 11, 2, 5]]
six = [0, 2, [9, 0, 4]]
seven = [0, 2, [11, 2, 5]]
progression = one
for i = 1 to 6
    progression = sc_append(progression, pick_next())
progression = sc_append(sc_append(progression, one), one)
score = sc_append(progression, progression)
```

Example: Influence melody with chord tones

- Imagine a set of prior probabilities for chosing a pitch class at time b: `prior[i]`
- Given a chord score `s`, let's make chord-tones twice as likely:

```
var pcs = sc_pitches_at(s, b)
for pc in pcs
    prior[pc] = prior[pc] * 2
```

- Pick a pitch class:

```
var pc = index_choice(prior)
```

Summary

- Markov Models
  - Easy to learn from examples
  - Only very local context
- Grammars
  - Recursive
  - Can generate concurrent structures
  - (Mostly) very local context
- Patterns
  - Expressive way to create abstract hierarchical structure
- Structure + Probability example
  - for popular music
  - Music production (instrumentation, texture, “arrangement”) is lacking

Summary 2

- "Algorithmic Music" (Markov, Grammars, Patterns, etc.)
  - Creates very interesting, specific music material
  - Often one develops a new "algorithm" or algorithmic materials for each composition
  - Strong impact on artistic thinking, 20th-21st C.
- AI techniques
  - More general,
  - Too homogeneous to be really interesting (IMO)
  - Catching popular and researcher's imagination

57

Spring 2019

Suffix Trees and Music

- Markov Chains use fixed number of previous states to determine probability of next state
- Standard implementation is a (sparse) matrix
- What if you could consider prefixes of length 1, 2, 3, … N for a fairly large N?
- Suffix tries and trees: fast access to next states given previous states

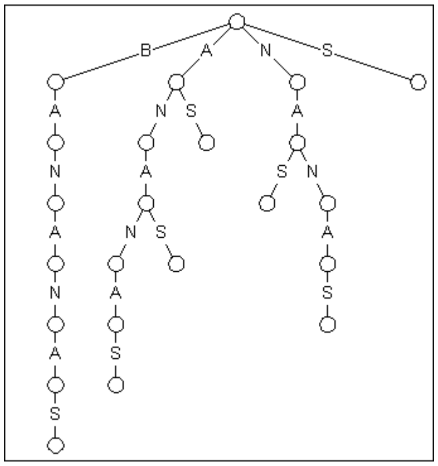
58

Spring 2019

What’s a Trie?

- See Wikipedia for an excellent overview
- An ordered tree structure
- Useful as an associative array
- Keys are strings
- Whole keys are not stored;
- Instead, key is a path from the root of the trie
- “Trie” from re*trie*val, pronounced either “tree” or “try” (I’ll use “try”).

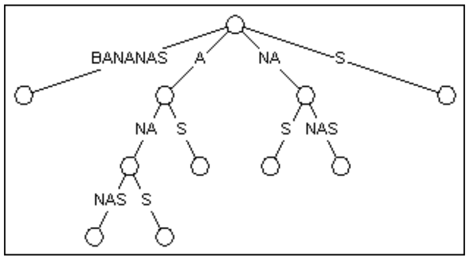
Suffix Trie



(from http://www.dogma.net/markn/articles/suffixt/suffixt.htm)

Suffix Tree

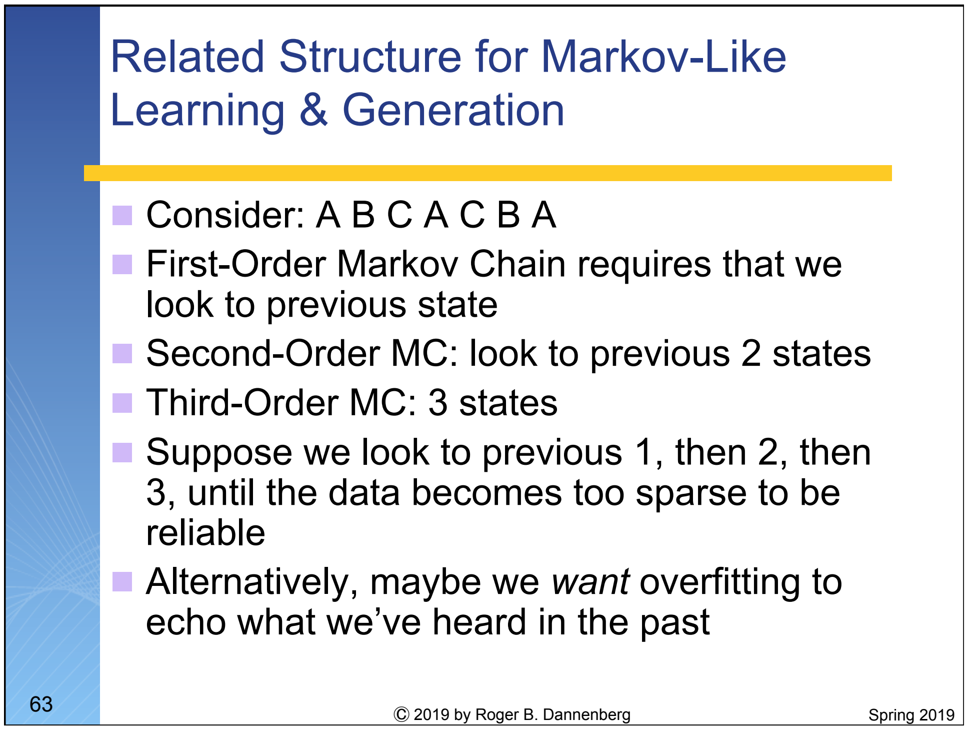
- Eliminate nodes with single descendent
- Represent nodes as <start, stop> index pair



(from http://www.dogma.net/markn/articles/suffixt/suffixt.htm)

Why Suffix Trees?

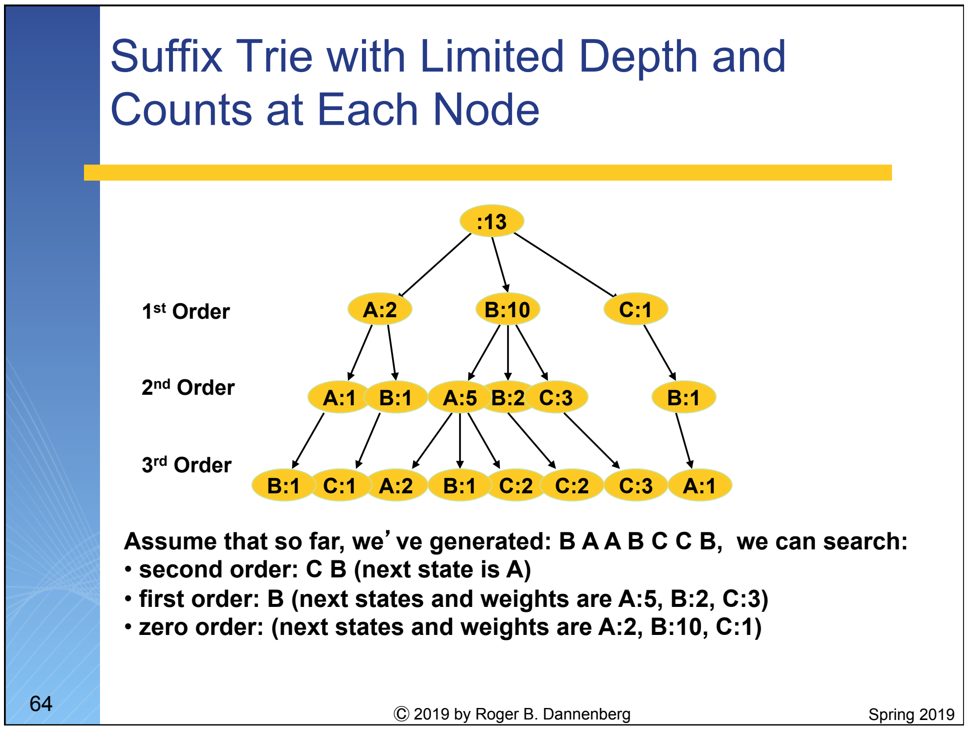
- Allows fast search for pattern in string:
  - O(n) preprocessing, where n is length of string
    - Note: the tree construction is non-trivial. Naïve construction is O(n^2).
  - O(m) per pattern search, where m is length of pattern



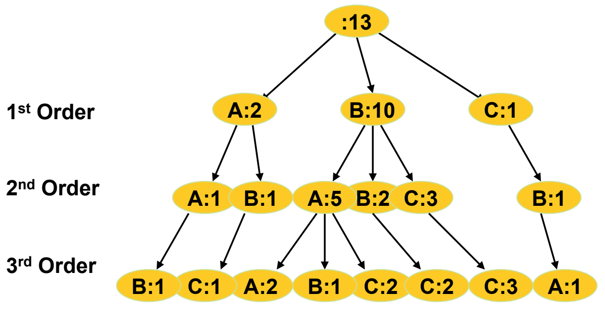
Related Structure for Markov-Like Learning & Generation

- Consider: A B C A C B A
- First-Order Markov Chain requires that we look to previous state
- Second-Order MC: look to previous 2 states
- Third-Order MC: 3 states
- Suppose we look to previous 1, then 2, then 3, until the data becomes too sparse to be reliable
- Alternatively, maybe we *want* overfitting to echo what we've heard in the past

63 Spring 2019



Suffix Trie with Limited Depth and Counts at Each Node



$$\begin{tikzcd}
 & & & & & {:}13 \arrow[dlll] \arrow[d] \arrow[drr] & & & \\
\text{1st Order} & & A{:}2 \arrow[dl] \arrow[d] & & & B{:}10 \arrow[dl] \arrow[d] \arrow[dr] & & C{:}1 \arrow[dr] & \\
\text{2nd Order} & A{:}1 \arrow[d] & B{:}1 \arrow[d] & & A{:}5 \arrow[dl] \arrow[d] \arrow[dr] & B{:}2 \arrow[dr] & C{:}3 \arrow[dr] & & B{:}1 \arrow[d] \\
\text{3rd Order} & B{:}1 & C{:}1 & A{:}2 & B{:}1 & C{:}2 & C{:}2 & C{:}3 & A{:}1
\end{tikzcd}$$

Assume that so far, we' ve generated: B A A B C C B, we can search:
- **second order: C B (next state is A)**
- **first order: B (next states and weights are A:5, B:2, C:3)**
- **zero order: (next states and weights are A:2, B:10, C:1)**

64 Spring 2019

Pruning the Tree

- Defn: Empirical probability
  - the number of times pattern appears divided by number of times it could possibly appear.
  - E.g. in "aabaaab", P("aa") = 3/6 = 0.5
- "Benefit of Context"
  - The empirical conditional probability is greater (by some factor) when the context is longer
  - E.g. P("b"|"aa") = 2/3, P("b"|"a") = 2/5; The ratio is 5/3 (the benefit of knowing "aa" vs. "a")

Based on: Dubnov, Assayag, Lartillot, Bejerano. "Using Machine-Learning Methods for Musical Style Modeling." IEEE Computer, August 2003.

Pruning the Tree (2)

- So tree retains only nodes where:
  - Pattern length < L
  - Empirical Probability > Pmin
  - Benefit of Context > r
- Smoothing: combine probabilities based on all matching patterns.
  - E.g. the next symbol x after "aabc" would combine P(x | "aabc"), P(x | "abc"), P(x | "bc"), P(x | "c") and P(x), omitting P's where context is not in the pruned tree.

Example

- Piano improvisation using variable order Markov Chain
- Analysis:
  - Reduce polyphony to sequence of “compound events”
  - States are (pitch class sets) x (log duration). [$2^{12} \cdot 5$ states]
    - 0 if <0.1, 1 if <0.2, 2 if <0.4, 3 if <0.8, 4 if >0.8
  - Create transition counts tables for 1st and 2nd order Markov Chains, using 12 different transpositions of the input data
  - Remember “real” performances (durations, velocity) for each state
- Generation:
  - Using the last state or last 2 states depending on choices and mode.
  - Pick a next state
  - Append a “real” performance of that state.

More Examples

- http://www.ircam.fr/equipes/repmus/MachineImpro
- **Example 1**
  - **1.1 Original improvisation by Chick Corea** Listen to Corea (mp3)
  - **1.2 Three machine improvisations generated after learning 1.1**
  - Listen to Impro 1 (mp3) Listen to Impro 2 (mp3) Listen to Impro 3 (mp3)
- **Example 2**
  - **One machine improvisation generated on "Donna Lee" by Charlie Parker**
  - Listen to Impro (mp3)

  Comment : **From a midifile containing an arrangement of this standard (theme exposition plus chorus). Took only the sax and bass channels. The strange bass rhythm behavior is due to a bug in the quantization algorithm, we kept it because the somewhat free style that results in an interesting remainder of some jazz tendencies in the sixties.**

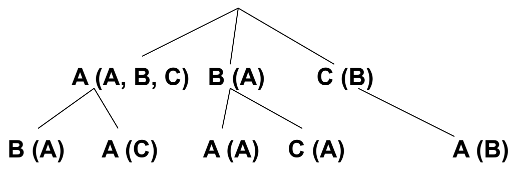
  The machine impro begins with a recombinant variant of the theme, then dives into a bop style chorus.
- **Example 3**
  - **One machine improvisation generated after learning J.S. Bach Ricercar**
  - Listen to Impro (mp3)
  - ***Comment :*** **Bach's ricercar is a six voice fugue. The information is extremely constrained, so the analysis/generation algorithm has very few choices for continuations. It tends to reproduce the original. But if you listen carefully, you'll hear that there are discrete bifurcations where it recombines differently from the original.**

Examples (2)

- Example 4
  - A study in the style of Jazz guitarist Pat Martino. Here's an idea of the original style (Blue Bossa) :
  - Listen to Pat Martino (mp3)
  - The learning process was based on a Midifile containing a transcription of Martino chorusing on Blue Bossa. After generating a few machine choruses, and choosing carefully a one that would fit, we mixed it back into Martino's audio recording, in a place where only the rhythmic section was playing (plus some piano). The machine impro is played with an (ugly) synthetic Midi Sax sound.
  - Listen to Mix (mp3)
  - *Comment :* That experience was done in order to evaluate if the techniques used could make sense in a performance situation, with a musician playing with his clone. The result is encourageing, but in a real-time experiment, we would have to extract the beat and the harmony in order to control what's happening. In this case, we just inserted the machine impro by hand, tuning the tempo so it would fit with the audio.
- Example 5
  - A Real-Time performance experiment.
  - Because the rhythm section is generated, we know the beat/harmony segmentation. The machine learns the correlation between the beat structure, the harmonic structure, and what's played by the performer. Sequence 5.1.
  - Listen to Sequence 5.1 (human on piano)
  - Listen to Sequence 5.2 (human + computer)
  - Listen to Sequence 5.3 (human’s new chords reused by computer)
  - Listen to Sequence 5.4 (computer alone)

Another Data Structure

- Paths from root to leaf nodes are *reverse* suffixes, e.g. for A B A A C B A,
  - A → B, B→C, AB→C, C→A, BC→A, ABC→A,



David Cope: Recombinant Music

- Create fragments from compositions
- Reassemble fragments to form pieces

- Search for patterns based on melodic intervals
- Harmonic context (chord progressions) of each melodic fragment are retained
- Patterns of harmony are also discovered

Signatures

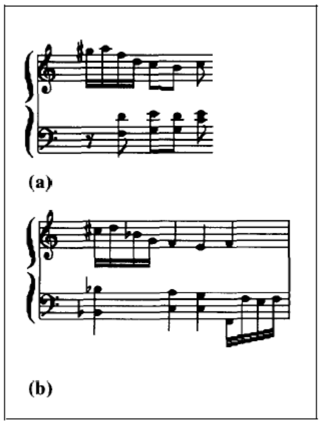


Figure 4. Two versions of a Mozart signature from (a) K. 330 and (b) K. 547a.

Recombination

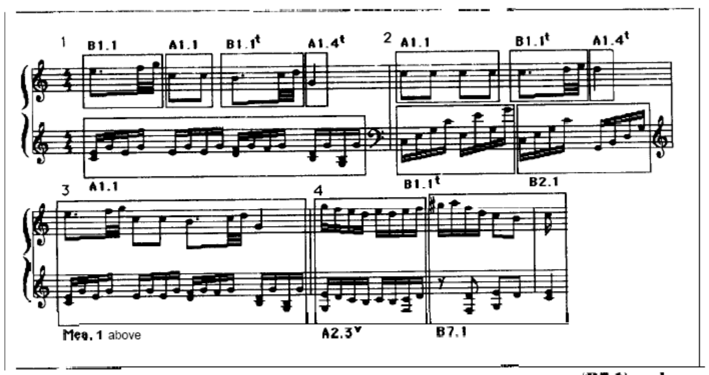


Figure 6. EMI's recombination of segments in Figure 5, with signature (87.1) and suggested sources (t = transposition; v = variation).

Examples

- Based on Scarlatti
- Based on Bach Invention
- Based on Joplin Rag

Recent Work

- Some interesting work on treating digital audio samples as learnable sequences:

Credit: WaveNet project from DeepMind
https://deepmind.com/blog/wavenet-generative-model-raw-audio/

- We can look at notes, chords, or other music representations in terms of sequences:

Credit: Sony CSL FlowComposer Project, 2016
http://www.flow-machines.com/wp-content/uploads/2016/06/Miles-Davis-Mix_DEF.mp3

Summary

- Sequence Learning can be applied to Music generation
- "symbols" can be pitches or, more likely, combinations of pitch+duration
- Markov Chain concepts can be extended to variable length suffixes
- Suffix trees and related structures provide efficient representations
- "Modern" machine learning approaches are actively (re)exploring these concepts