A Tutorial on Pattern Discovery in Symbolic Music Information Retrieval

Ian Knopke

BBC, Future Media and Technology, ian.knopke@gmail.com

August 9, 2010

Outline

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. Filtering Results
9. Approximate Matching
   - Edit Distance
10. Additional Topics

Layout

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. Filtering Results
9. Approximate Matching
   - Edit Distance
10. Additional Topics

Introduction

Pattern discovery and search is about finding patterns in musical data.

There is quite a bit of literature out there on the topic.

This tutorial is primarily concerned with *melodic* patterns.

Introduction

My primary research interest is in systems for working with large collections of music.

I see these as primarily a musicological / analytical research areas, although it also has compositional and commercial applications

These technologies can be used to provide new directions for musicology and music theory (provided one can find a way to ask the right questions)

They also have potential for applications such as building search engines

Issues

- Single piece (intra-opus)
- multiple pieces (inter-opus)
- exact matching vs approximate matching
- known patterns vs unknown patterns

Layout

1 Introduction

2 Basic Symbolic Concepts
- A Brief Introduction to Humdrum

3 Exact Matching

4 Suffix-Based Approaches

5 Mappings

6 Case Study: Palestrina

7 Multidimensionality

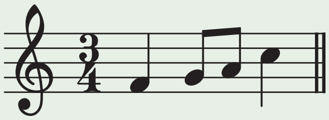
8 Filtering Results

9 Approximate Matching
- Edit Distance

10 Additional Topics

The Staff

Example



The musical staff is a type of specialized graphing system (that precedes Descarte!)

The time axis proceeds to the left.

The pitch axis is centered through the use of clefs.

Collections of staves form a score.

This is the basic “canvas” of western music.

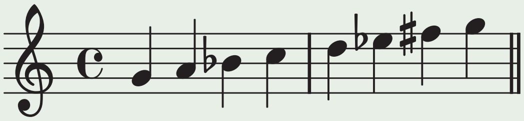
Pitches

Pitches are measurements over the frequency space.

For historical reasons, the pitch axis is slightly warped and doesn't leave places for all notes.

These are modified using accidentals.

Scale with Accidentals



Durations are measurements over time, given as subdivisions of a beat.

Intervals

Intervals are a distance between two measurements.

- It takes two elements to make one interval.
- Interval lists are always one shorter than a sequence of pitches.
- Essentially a kind of discrete differentiation over a measure

Intervals are often preferred because they are transposition-invariant.

- Pitch intervals are the difference between successive pitches (in semitones).
- Duration intervals are the division between successive durations (in beats).

Melody and Harmony

A melody consists of successive pitches in a mostly-linear fashion over time.

Harmony happens when different pitches are sounded simultaneously.

Simultaneous sets of two notes are called harmonic intervals or dyads.
Proper harmony is based on the triad and requires at least three notes to form a chord.

Additional Musical Elements

Most of the time in symbolic music processing we concentrate on pitch and duration.

Some other elements, often ignored, that can be symbolically encoded:

- Articulations
- Dynamic markings
- Beat
- Score directions

Many other types possible.

What's not in this model

Primarily talking about western or common music notation here. This is an abstraction where sonic concepts are represented by symbols.
Many elements are not included here:

- Timbre (except as orchestration)
- Tuning
- Notes that change over time (except crescendo)
- Non-western elements
- Many performance characteristics

Symbolic Representation Systems

These are just a few of the most popular ones. There are others...

- MIDI
- MusicXML
- Plain and Easie
- abc
- MEI
- Humdrum

Comprehensive listing: Beyond MIDI: The Handbook of Musical Codes, edited by Eleanor Selfridge-Field, 1997.

Layout

1. Introduction
2. **Basic Symbolic Concepts**
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. Filtering Results
9. Approximate Matching
   - Edit Distance
10. Additional Topics

So what is Humdrum?

Humdrum toolkit originally by David Huron in the mid 90s.

Humdrum is designed for the analysis of music in symbolic form.

Consists of two parts: the data format, and the accompanying tool collection.

Data format is excellent, especially for music analysis tasks.

Example: Humdrum

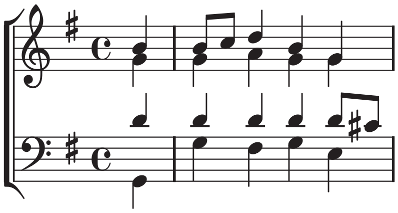


Figure: *Uns ist ein Kindlein heut' gebor'n*, J. S. Bach

Example: Humdrum

```
**kern  **kern  **kern  **kern
*bass   *tenor  *alto   *soprn
*k[f#]  *k[f#]  *k[f#]  *k[f#]
4GG     4d      4g      4b#
=1      =1      =1      =1
4G      4d      4g      8b#
.       .       .       8cc
4F#     4d      4a      4dd
4G      4d      4g      4b
4E      8d      4g      4g
.       8c#     .       .
```

Humdrum

- Horizontal staves are encoded as vertical *spines* (90 degree rotation).
- Notes encoded as letters. Middle c (261.63 Hz) is encoded as “c”;
- Octaves, from low to high: (CCC,CC,C,c,cc,ccc, etc.)
- Durations are encoded as numeric values (eighth note is 8, etc.)
- Tokens with asterisks are *interpretations*.
- Most common format is `**Kern` format.

Original tools are all command line (CLI).

What Humdrum is not

Humdrum is not a notation format, and probably isn’t the best choice for archiving.

The primary use of Humdrum is for analyzing symbolic music using computers.

Both the tools and the data format are necessary parts of that process.

David Huron gave this problem a lot of thought, and the design of the system is a reflection of this.

MIDI, MusicXML, or other storage formats may replace the data, but not the tools.

Layout

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. **Exact Matching**
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. Filtering Results
9. Approximate Matching
   - Edit Distance
10. Additional Topics

Exact and Approximate Matching

Searching for exact matches is often called pattern matching.

Searching for matches with various degrees of variation is called approximate pattern matching or inexact matching.

Wikipedia says:

"In computer science, pattern matching is the act of checking some sequence of tokens for the presence of the constituents of some pattern. In contrast to pattern recognition, the match usually has to be exact"

The Simplest Search Algorithm

Naive Interleaved String Match

```
I  := \emptyset
\FOR j:=0 \TO n-m \DO
    i=1
    \WHILE p_{i} \eq t_{j+i} \textrm{ and } i \leq
        i:=i+1
    \IF i=m+1
    \THEN
        I:=I \cup {j+1}
```

(from "Algorithmic Aspects of Bioinformatics", 2008, Bockenhauer et al.)

Naive Match

This is an extremely intuitive algorithm!

Matching pattern (abc) against str (abdadabc)

```
a b d a d a b c

a b X
  X
    X
      a X
        X
          a b c   MATCH FOUND
```

Others

Exact string matching is an extremely well-studied problem!

Some examples include:

- Knuth-Morris-Pratt
- Boyer-Moore
- Karp-Rabin
- Baeza-Yates-Gonnet

Many other algorithms,

Most work by preprocessing the search pattern to reduce the number of tests to be done.

How useful are these with music?

Most of these other algorithms are more efficient than the naive algorithm

However, they (should) all produce the same results

Modern computers are very fast

Consider: music is multidimensional

Searching on a single attribute (pitch, duration) usually produces many false positives.

it can take a lot of time to adapt these other algorithms to more than one dimension

(I'll show a quick way to do this later)

Layout

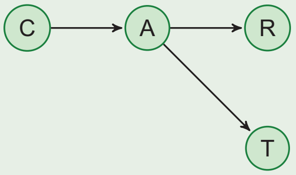
1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. **Suffix-Based Approaches**
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. Filtering Results
9. Approximate Matching
   - Edit Distance
10. Additional Topics

Trie

A trie is a multikey tree structure for comparitively storing multiple patterns

Each path from the root to a leaf represents a different string from a collection.

Example



New nodes are created at the points where two strings differ. e.g., car and cat branch on the third letter

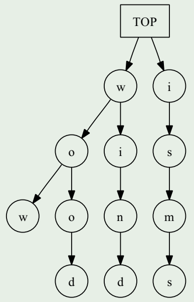
Tries are very fast to check for existence of strings.

Commonly used for lookup dictionaries (attach index to leaf).

The name comes from information retrieval.

Trie - 1

Trie



A suffix trie for the terms {wow, wood, wind, isms}.

Prefix, Suffix

A prefix of a string is a substring of length *n* beginning at the first index:

Prefix

$\text{prefix}(x) = x[1..j], j = \text{int}$

`b a n` is a prefix of `b a n a n a`

A suffix of a string is a substring of length *n* ending at the last index:

Suffix

$\text{suffix}(x) = x[j..n], j = \text{int}$

`a n a n a` is a suffix of `b a n a n a`

Suffix Trie - 1

A very clever structure that takes successive suffixes of a string and inserts them into a trie structure.

Suffixes of b a n a n a s

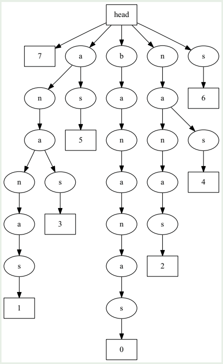
Indices	Substring
$x_0..x_7$	b a n a n a s ϵ
$x_1..x_7$	a n a n a s ϵ
$x_2..x_7$	n a n a s ϵ
$x_3..x_7$	a n a s ϵ
$x_4..x_7$	n a s ϵ
$x_5..x_7$	a s ϵ
$x_6..x_7$	s ϵ
x_7	ϵ

Requires a unique end character to be assigned or we lose branches.

The position of each suffix is stored as the end node of each

Suffix Trie - 2

Suffix Trie

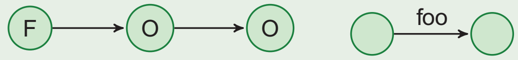


Compact Suffix Trie

Problem with the Suffix Trie is that it all those nodes take a lot of memory

More compact version is called a compact suffix trie, a Patricia (PAT) tree, or a Suffix Tree

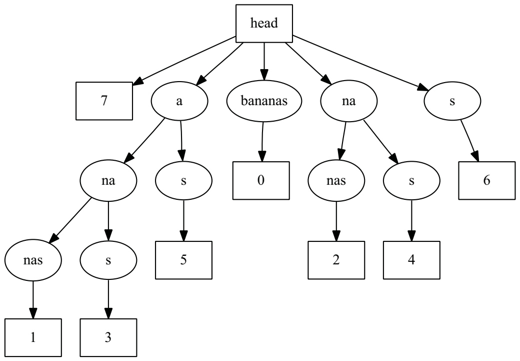
Example



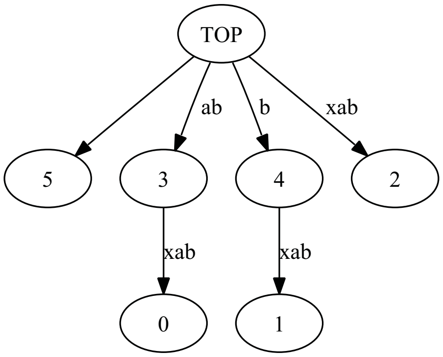
Basically it compacts sets of nodes without branches into trees again.
Easy to construct from an existing suffix trie.

There are also very good online construction algorithms (Ukkonen 1995, Farach 1997)

Suffix Tree



Another Suffix Tree



What do they solve?

The suffix tree is one of the most important data structures in string matching.

- Quickly find all exact matches in a data set
- Longest Common Substring Problem
- Many other problems in string searching
- “Exotic searches” such as palindromes
- Other varieties: DAWG, Factor Oracle

Problems

- Basic Problem: uses a great deal of memory
- Usually built in memory, but this is finite.
- Disk-based trees with multiple branches are tricky to implement (this is basically a database)
- Trees have an unpredictable number of branches.
- Programmer time (ie ME) is worth something

Suffix Array - Advantages

An alternative to suffix trees

- Disk based, requiring only 4n bytes.
- Can be constructed in O(n) time.
- Conceptually much simpler to program than disk-based trees.
- Somewhat easier to display results (and debug).
- Some advantages for approximate searching (later...)
- If the length of suffixes is windowed this is effectively a kind of *n-gram*.
- Basically an enumeration of all of the root-leaf paths of the suffix tree.

Suffix Array Example - 1

Start with the sequence of suffixes of a string, listed incrementally.

Suffix Array

Indices	Substring
$x_0..x_7$	`b a n a n a s` ϵ
$x_1..x_7$	`a n a n a s` ϵ
$x_2..x_7$	`n a n a s` ϵ
$x_3..x_7$	`a n a s` ϵ
$x_4..x_7$	`n a s` ϵ
$x_5..x_7$	`a s` ϵ
$x_6..x_7$	`s` ϵ
x_7	ϵ

Suffix Array Example - 1

Then sort the strings.

Suffix Array

Indices	Substring
$x_1..x_7$	`a n a n a s` ϵ
$x_3..x_7$	`a n a s` ϵ
$x_5..x_7$	`a s` ϵ
$x_0..x_7$	`b a n a n a s` ϵ
$x_2..x_7$	`n a n a s` ϵ
$x_4..x_7$	`n a s` ϵ
$x_6..x_7$	`s` ϵ
x_7	ϵ

We have excellent disk-based sort algorithms, including UNIX `sort`.

Disk-based, so this can scale to extremely large data sizes.

Layout

1 Introduction

2 Basic Symbolic Concepts
- A Brief Introduction to Humdrum

3 Exact Matching

4 Suffix-Based Approaches

5 Mappings

6 Case Study: Palestrina

7 Multidimensionality

8 Filtering Results

9 Approximate Matching
- Edit Distance

10 Additional Topics

Mapping Music to Chars

Need single characters for text sorting

Didn’t want to program a whole new sort algorithm for music tokens

Instead, I build a list of tokens (alphabet) and map these to ASCII characters

It’s a good idea to understand how the lexicographical sort mechanism works for your algorithm

Do numbers come before capitals, punctuation, numbers, etc.?

Mapping Music to Chars - 2

Mapping

```
     ’0’  => ’a’,
     ’1’  => ’b’,
     ’2’  => ’c’,
     ’3’  => ’d’,
     ’4’  => ’e’,
     ’5’  => ’f’
     ’6’  => ’g’,
     ’7’  => ’h’,
     ’8’  => ’i’,
     ’9’  => ’j’,
     ’10’  => ’k’,
     ’11’  => ’l’
```

Mapping Music to Chars - 3

Another mapping:

Pitch-Mapping

```
’a’  => ’d’,
’a#’  => ’k’,
’b’  => ’h’,
’c’  => ’g’,
’c#’  => ’l’,
’d’  => ’e’,
’d#’  => ’b’,
’e’  => ’a’,
’f’  => ’m’
’f#’  => ’f’,
’g’  => ’i’,
’g#’  => ’j’,
’r’  => ’c’,
```

Mapping Music to Chars - 4

About 96 characters are useable from the basic ASCII 0-127 set

Characters I can't see (like spaces and the bell) are excluded for now

For larger sets, can use full ASCII set (255 chars)

For even larger sets, can use Unicode (about 40,000 chars)

Metadata Database

- Each line in the array file is given an index
- Stored using the tilde, the last sort character
- PostgreSQL entry for every note in Palestrina
- Stores original file, spine, line number, original token, mapped character
- Later added information for retrogrades and inversions

What it looks like (unsorted)

Unsorted

```
dafpupmludamadd~29
afpupmludamadd~30
fpupmludamadd~31
pupmludamadd~32
upmludamadd~33
pmludamadd~34
mludamadd~35
ludamadd~36
udamadd~37
damadd~38
amadd~39
madd~40
add~41
dd~42
d~43
```

What it looks like (sorted)

Sorted

```
aamlluddadpmp~1723639
aamll~194467
aamlpddppisdad~1130546
aamlpmedp~694903
aamlpmegqdpppmedpl~1150398
aampmad~1718834
aampmppiuigldpmppl~711986
aamppimppplmll~1558198
aamrdddamaipmpdaqp~1723605
aamrdddaqam~34555
aam~1141189
aam~1150394
```

Layout

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. **Case Study: Palestrina**
7. Multidimensionality
8. Filtering Results
9. Approximate Matching
   - Edit Distance
10. Additional Topics

Overview

Palestrina

Giovanni Pierluigi da Palestrina (1525)

Musicology

Palestrina was a composer and master of counterpoint

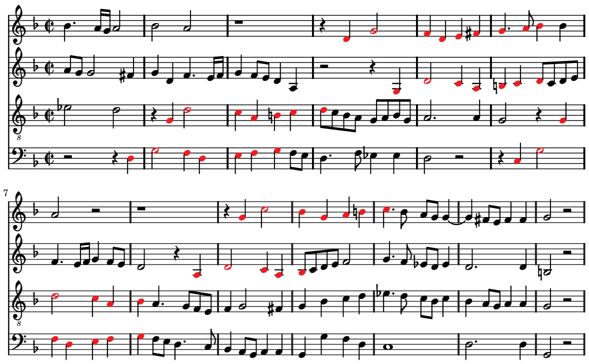
In the interests of a particular musicological question, we wished to find common melodic matches across his entire corpus of 104 masses

Exact search techniques won’t work because we don’t know any patterns yet!

Used a suffix array approach

All processing done using the PerlHumdrum Analysis Toolkit (PHAT)

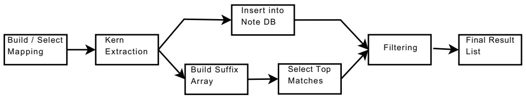
Contrapuntal Example



Palestrina Statistics

- 104 masses
- 1315 Mass parts (Gloria, Kyrie, etc.)
- 1,706,027 tokens
- 801,779 pitches (not including ties)
- 709,592 intervals
- longest matches are canons in same pieces

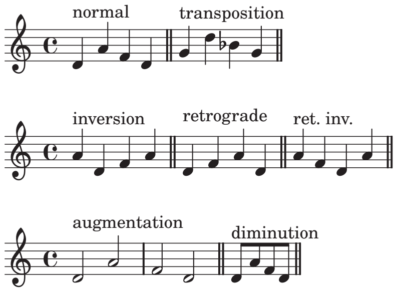
System



Processing Steps

1. PerlHumdrum extraction from Humdrum files
2. Extract a token list
3. Mapping music tokens to single characters
4. Build a suffix array and PostgreSQL database
5. Sort the suffix array file
6. Search for results

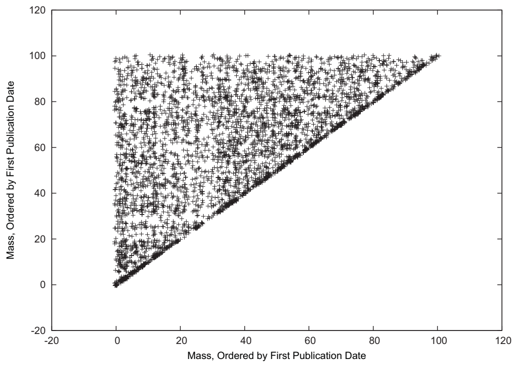
Melodic Possibilities



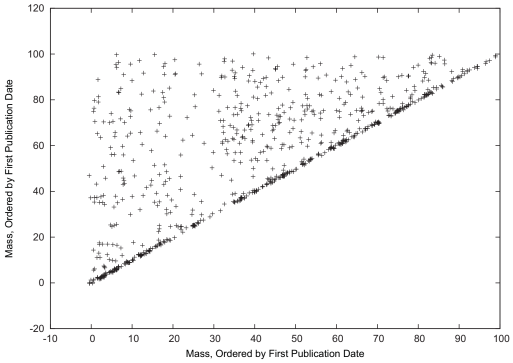
Search Types

Pitch Interval	transposition, inversion
Duration Interval	retrograde, augmentation, diminution
Combined	retrograde inversion, other

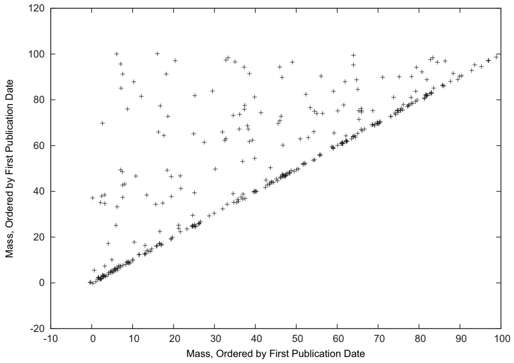
Matches (len 6+)



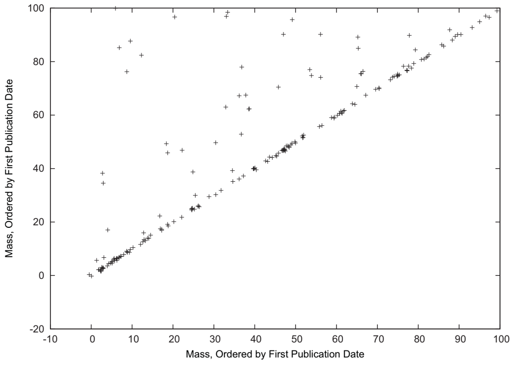
Matches (len 8+)



Matches (len 9+)



Matches (len 10+)



Layout

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. **Multidimensionality**
8. Filtering Results
9. Approximate Matching
   - Edit Distance
10. Additional Topics

The problem

- Pattern matching in strings is a very active area of research right now
- However, sequence matching on music raises problems unique to the field
- Unlike other areas, music is inherently multidimensional (pitch, rhythm, etc.)
- Additionally, this doesn't fit well into the string/sequence paradigm, which is inherently single character/symbol
- This seems to be a recurring problem in this research
- Also shows up in many MIR and computer music problems in general

Approaches to multidimensionality

- Use a single dimension (usually pitch) and ignore others (most common)
- Use multiple dimensions separately, and try to coordinate
- Cartesian product
- Conklin's "Multiple Viewpoints"
- Statistical methods (PCA, MDS)
- Use a classifier to map multiple dimensions to single (HMM).

Cartesian Product

- Previously used a Cartesian product approach across a large data set.
- Each token was a combination of pitch and rhythmic interval
- Data set had over 800,000 notes
- Generated several hundred combined tokens that were used!
- These tokens were then remapped to an ordered list of single characters
- This was much too large for ASCII
- Used Unicode (UTF-8) instead

Cartesian Product

Cartesian Product

$$t : p \times d \to \Sigma_U, \text{ where } p \in \Sigma_{pitch}, d \in \Sigma_{dur} \qquad (1)$$

Pitch - Rhythm Mapping

	1/2	1/4	1/8
b	i	j	k
c	l	m	n

Pitches	[b,c]
Durations	[1/2,1/4,1/8]
Σ	[i,j,k,l,m,n] (UTF-8)

Why I don't do that anymore

This worked. However. . .

- Creates a very large token set
- Tokens have no syntactic relation
- Difficult to check results with (some unicode not visible)
- Sort order issues
- Not a "musical" solution

New Approach

The old single-token Cartesian product:

Example

(t) = token, where:
$t : p \times d \rightarrow \Sigma_U$
p=pitch,d=duration

New approach is an ordered tuple I call an interleaved melodic token:

Example

(p, d) = (pitch,duration), where:
$p \subseteq \Sigma_P i, j \subseteq \Sigma_P, i < j$
$d \subseteq \Sigma_D k, l \subseteq \Sigma_D, k < l$
$\Sigma_P < \Sigma_D$
$\Sigma_P \cap \Sigma_D = \{\emptyset\}$
$\Sigma_P, \Sigma_D \subseteq \Sigma_U$

Interleaved Melodic Encoding

An interleaved melodic encoding is an ordered tuple of two symbols, pitch(p) and duration(d)

p and d are drawn from different alphabets Σ_P, Σ_D

Each alphabet has a defined lexicographical ordering

Each alphabet is also a subset of the same larger alphabet

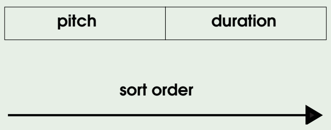
This alphabet also has a sort order.

Interleaved Melodic Encoding

In other words, every pitch and duration is mapped to a single character.

However, every pitch token will sort earlier than every duration token.

Interleaved



(Could also put durations earlier in sort order ahead of pitches - it doesn't matter)

Encoding Example

Cartesian Product

	f	g	a	c
1/4	s	t	u	v
1/8	w	x	y	z

Pitches	[f,g,a,c]
Durations	[1/4,1/8]
Σ	[s,t,u,v,w,x,y,z] (UTF-8)

Interleaved Melodic Encoding

Pitches	[f,g,a,c]	[f,g,a,c]
Durations	[1/4,1/8]	[x,y]
Σ	[f,g,a,c,x,y]	

Why is this important?

There is a very practical interpretation of all this . . .

A regular expression is a formal system for describing the collection of strings that comprise a language.

Often used to define symbolic matches, including approximate matches in text sequences

Regular expressions are considered to be equivalent to automata (type 3 grammar) as defined in the Chomsky language hierarchy.

Many languages, such as Perl, have very efficient regular expression engines.

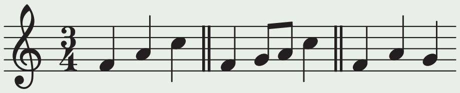
Also efficient programs such as Unix *grep*.

Regular Expressions

Interleaved Melodic Encoding

Pitches	[f,g,a,c]	[f,g,a,c]
Durations	[1/4,1/8]	[x,y]
Σ	[f,g,a,c,x,y]	

Example

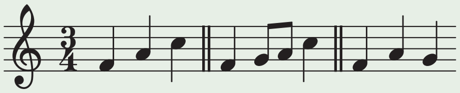


Encoding

```
fx ax cx fx | gy ay cx | fx ax gx
fxaxcxfxgyaycxfxaxgx
```

Regular Expressions

Example



Single token:	/fx/
Arpeggio:	/fxaxcx/
Two Melodies (NFA)	/fxaxcx|fxaxgx/
Two Melodies (DFA)	/fxax(c|g)x/
Three Melodies	/fxaxcx|fxgyaycx|fxaxgx/
	/fxaxcx|fxgyaycx|fxaxgx/
	/fx(axcx|gyaycx|axgx)/
	/fx((ax|gyay)cx|axgx)/
f a pitches, any dur	/f.a.*/
quarter notes only	/(.x)+/

Approximate matching, wildcards and alphabets

Approximate matching is possible through the use of the *Kleene star*(*) and it's variations.

Example

*	accept any number of pitches or durations, including none
+	accept one or more pitches or durations
?	accept one or none

It's also possible to restrict matches to particular alphabets, or subsets:

Example

/[fgac]x/	match any 1/4 note
/[fac][xy]/	match any arpeggio note with 1/4 or 1/8 dur

The exclusivity of the alphabets ensures there aren't cross matches.

The Simplest Approximate Search Engine Ever

. . . for symbolic music.

1. Find a collection of files in a symbolic format
2. Encode the music as a long string of interleaved melodic tokens
3. construct a search query as an interleaved melodic automaton
4. Convert to a regular expression as needed
5. Run the regex across the encoded string

Simple

Simple Engine

```
my $str='fxaxcxfxgyaycxfxaxgx';

while($str=~/(fxax.x)/ig){

    my $match=$1;   #actual string matched

    ##true location
    my $location=pos($str)- length($match);

}
```

Returns matches at 0,15.

This is really fast!

Additional Advantages of Interleaved Encodings

- It's easy to do single-dimension searches by skipping characters:
  - Even are pitch
  - Odd are durations
- Can be extended to include other features, such as articulations
- Token set is much smaller and can be precalculated
- Takes less space than unicode
- Interleaved melodic tokens work very well with suffix trees and suffix arrays
- Easy to drop into existing sequence search algorithms
- Extremely useful where variations are known (*musica ficta*)

Interleaved Melodic Encodings and Suffix Arrays

Interleaved melodic encodings work very well with suffix arrays.

- Overall sort order means that durations sort after pitches.
- Takes twice as many tokens, but may still be smaller than unicode.
- Certainly more readable!
- All of the strings starting with durations appear at the end

Suffix Array Example

Suffix Array (Unsorted)

```
fxaxcx~1
xaxcx~2
axcx~3
xcx~4
cx~5
x~6
fxaxgx~7
xaxgx~8
axgx~9
xgx~10
gx~11
x~12
```

(Only using the first and third phrases to save space)

Suffix Array Example

Suffix Array (Sorted)

```
axcx~3
axgx~17
cx~5
fxaxcx~1
fxaxgx~15
gx~19
xaxcx~2
xaxgx~16
xcx~4
xgx~18
x~20
x~6
```

Adapting Existing String Matching Algorithms

An Interleaved Encoding version of the naive string matching algorithm.

Naive Interleaved String Match

```
I := \emptyset
\FOR j:=0 \TO n-m \DO
    i=1
    \WHILE \textrm{compare}( p_{i},d_{i+1},p_{j+1},
        i:=i+2
    \IF i=m+2
    \THEN
        I:=I \cup {j+1}
```

Algorithms that work well with this

- Exact matching
- Palindrome detection
- Maximal pairs
- Supermaximal pairs
- Tandem repeats
- Edit distance
- Pairwise alignments
- Longest common subsequence / substring
- K-mismatch
- Suffix tree / array

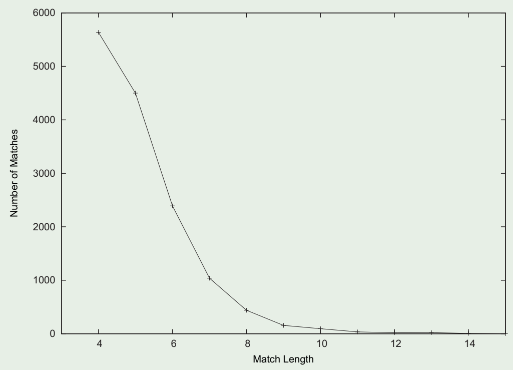
Lots of others I haven't tried yet.

Layout

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. **Filtering Results**
9. Approximate Matching
   - Edit Distance
10. Additional Topics

Match Lengths

Matches



Filtering Results

Musical pattern matching problems seem to suffer from generating too many results.

On the other hand, we often want to find approximate matches, also increasing the number of results.

This is a problem!

Ultimately the understanding as to what is a relevant musical result is probably going to come from better understanding of cognition.

In the meantime, here are some things you can try.

Filtering Methods

Interval Class

- M3,m3 = thirds
- works for major, minor versions of same melodies
- Other modal (dorian, mixolydian) possibilties
- Works well for jazz

Contour

- up, down, leap, unison
- Contour preservers melodic outlines
- Susceptible to comparing very different things sometimes

Filtering Methods

Pitch Class

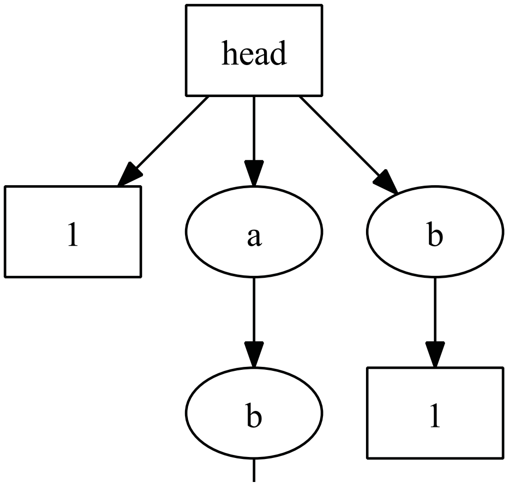
- 1=12, 2=11, 3=6, 4=5, 6=6
- Originally from set theory, Allan forte
- groups inversions together
- . . . but confuses changes in directions

Modal

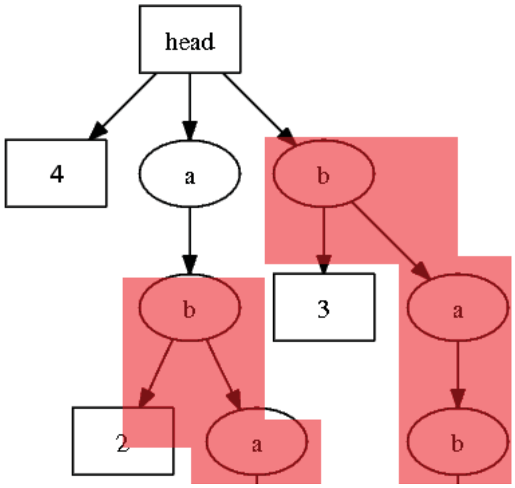
- In renaissance music, modal information can be used
- Takes aspects of mean-tone tuning or pythagorean tuning into account

Combinations are possible and desireable!

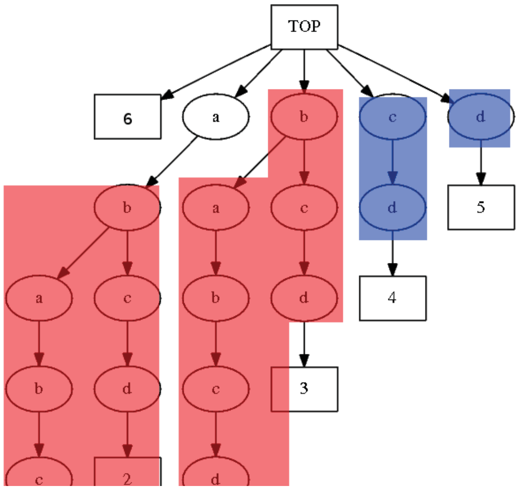
Suffix Tree Difficulties



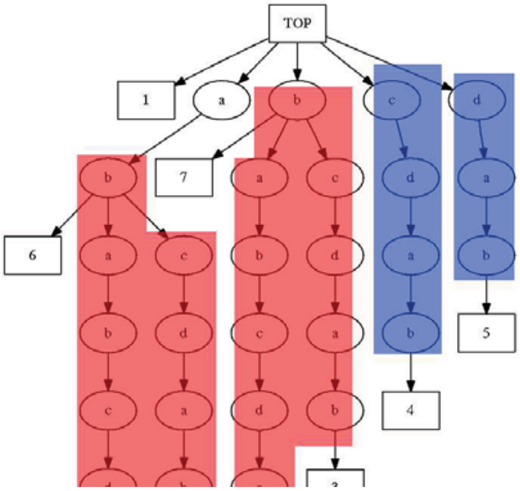
Suffix Tree Difficulties



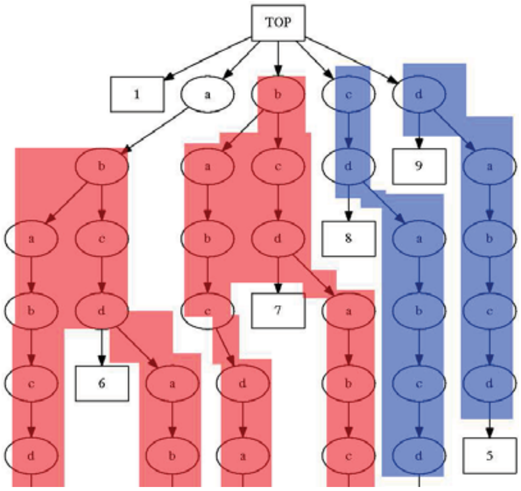
Suffix Tree Difficulties



Suffix Tree Difficulties



Suffix Tree Difficulties



Methods Specific to Suffix Trees

Maximal Repeat:

Supermaximal repeats:

All these methods require the calculation of the lowest common ancester

Also enhanced suffix arrays

Layout

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. Filtering Results
9. **Approximate Matching**
   - Edit Distance
10. Additional Topics

Sequence Alignment

Definition

The alignment of two sequences of symbols to be as similar as possible.

Alignment

```
C L A S P - -
J - A S P E R
```

This may require gaps to be inserted or elements removed from one of the sequences.

Layout

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. Filtering Results
9. **Approximate Matching**
   - Edit Distance
10. Additional Topics

Edit Distance

The *Edit Distance* is the minimum number of alterations necessary to transform one sequence of symbols into another.

Typically three operations are defined:

- Insertion (lean to learn)
- Deletion (chimp to chip)
- Replacment / Substitution (hare to mare)

It is also useful to define a match operation for situations where the two sequences already align.

Also known as Levenstein distance, Smith-Waterman, Needleman-Sunsch

Let’s Turn WATER into WINE

Example

		WATER
match	W to W	W A T E R
replace	A with I	W I T E R
replace	T with N	W I N E R
match	E to E	W I N E R
delete	R	W I N E

Alignment

```
W  A  T  E  R
W  I  N  E  -
```

Edit Distance

Edit distance is usually defined recursively:

Recursive

$$S(i,j) = \min \begin{cases} s(i,j-1) & + \ \omega_{insert}, \\ s(i-1,j) & + \ \omega_{delete}, \\ s(i-1,j-1) & + \ \omega_{replace} \end{cases}$$

where $\omega_{insert}, \omega_{delete}, \omega_{replace}$ are the costs associated with each operation.

In the most basic cases these are set to 1.

The sum of all operations gives a *similarity measure* between the two sequences.

Recursive Implementation

Recursive

```
sub ed{
    #1st str, 2nd str, 1st ind, 2nd ind
    my ($s,$t,$i,$j)=@_;

    ##initial insert cost
    return &insert() * $j if $i==0;
    ##initial delete cost
    return &delete() * $i if $j==0;

    return &min(
        &ed($s,$t,$i-1,$j)
            + &insert(substr($s,$i,1)),
        &ed($s,$t,$i,$j-1)
            + &delete(substr($t,$j,1)),
        &ed($s,$t,$i-1,$j-1)
            + &match(substr($s,$i,1),substr($t,$j,1
```

Dynamic Programming

- Very commonly used in bioinformatics for sequence alignment, similarity
- Also used in Hidden Markov Models (Viterbi algorithm)
- Many time-series problems exhibit these properties, including music problems
- Originally given this name by Richard Bellman (ref)
- Programming refers to finding an optimal *program* in the scheduling sense

Dynamic Programming

Dynamic programming is said to have two properties:

Bellman Equation

Optimal substructure: the solution can be obtained through the combination of parts of the original problem. Overlapping

subproblems: solutions to subproblems can be made from smaller versions of the same problem.

The second point is basically recursion.

The first implies some kind of caching might be in order . . .

Edit Distance

The recursive definition suggests an array representation for storing intermediate results.

DP Definition

$$\text{where } \omega_{replace} = 0 \text{ if same, else } \omega = 1 \qquad (2)$$

$$\begin{tikzcd}
s(i-1,j-1)+\omega_r \arrow[dr] & s(i,j-1)+\omega_d \arrow[d] \\
s(i-1,j)+\omega_d \arrow[r] & S(i,j)
\end{tikzcd}$$

DP Worked Example

WATER->WINE

-	-	W	I	N	E
-	0	1	2	3	4
W	1	.	.	.	.
A	2	.	.	.	.
T	3	.	.	.	.
E	4	.	.	.	.
R	5	.	.	.	.

Horizontal values represent the cost of *insertion*. Vertical

values represent the cost of *deletion*.

DP Worked Example

WATER->WINE

-	-	W	I	N	E
-	0	1	2	3	4
W	1	0	.	.	.
A	2	.	.	.	.
T	3	.	.	.	.
E	4	.	.	.	.
R	5	.	.	.	.

DP Worked Example

WATER->WINE

-	-	W	I	N	E
-	0	1	2	3	4
W	1	0	1	2	3
A	2	.	.	.	.
T	3	.	.	.	.
E	4	.	.	.	.
R	5	.	.	.	.

DP Worked Example

WATER->WINE

-	-	W	I	N	E
-	0	1	2	3	4
W	1	0	1	2	3
A	2	1	1	2	3
T	3	.	.	.	.
E	4	.	.	.	.
R	5	.	.	.	.

DP Worked Example

WATER->WINE

-	-	W	I	N	E
-	0	1	2	3	4
W	1	0	1	2	3
A	2	1	1	2	3
T	3	2	2	2	3
E	4	.	.	.	.
R	5	.	.	.	.

DP Worked Example

WATER->WINE

-	-	W	I	N	E
-	0	1	2	3	4
W	1	0	1	2	3
A	2	1	1	2	3
T	3	2	2	2	3
E	4	3	3	3	2
R	5	4	4	4	3

The edit distance between (WATER and WINE) is 3

Backtrace

Sometimes all we want is the score between two sequences.

The min function essentially chooses between the lowest value for the insertion, deletion, and replacement operations.

Recording which operation was chosen makes it possible to define the *backtrace* of the sequence.

This makes it possible to determine the actual alignment between the two strings.

WATER -> WINE

Backtrace

-	-	W	I	N	E
-	0	.	.	.	.
W	.	0	.	.	.
A	.	.	1	.	.
T	.	.	.	2	.
E	.	.	.	.	2
R	.	.	.	.	3

Alignment

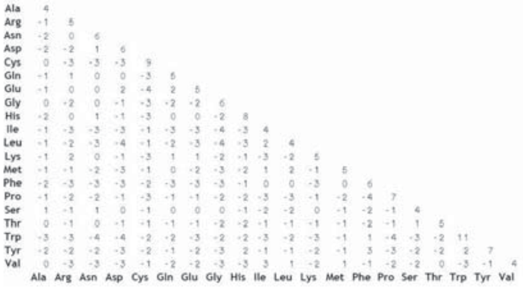
```
W A T E R
W I N E -
```

Weights and Biases

Better matches are obtained using a substitution matrix

The BLOSUM matrices are basically what makes bioinformatics possible.

Derived statistically from very large sets, so this approach may not work as well for music.



Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4
	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val

Other work

- Levenshtein
- Smith-Waterman
- Needleman-Wunsch
- Mongeau, and Sankoff, "Comparison of musical sequences", 1990.
- Smith, Mcnab, Witten, "Sequence-Based Melodic Comparison: A Dynamic-Programming Approach", 1998.

Peter van Kranenburg and Frans Wiering as really investigated this. Read Peter's dissertation!

Layout

1. Introduction
2. Basic Symbolic Concepts
   - A Brief Introduction to Humdrum
3. Exact Matching
4. Suffix-Based Approaches
5. Mappings
6. Case Study: Palestrina
7. Multidimensionality
8. Filtering Results
9. Approximate Matching
   - Edit Distance
10. **Additional Topics**

Geometric Approaches

Actually treats the staff as a graphing system!

- Defines notes as points in space/time
- Distance is determined through cosine similarity
- Computationally expensive, doesn't scale well
- Also needs lots of filtering
- However, may deal with polyphony better than string-based techniques

See Meredith, Lemstrom, Wiggins.

Other

- Conklin: Multiple Viewpoints
- Cambouropolis Bregman-based methods
- Factor Oracle
- Probabilistic Methods
- Tree-based edit distance (Rizo)

Questions!

Ian Knopke ian.knopke@gmail.com

Cognitive Approaches to MIR

Ian Knopke, BBC

Eric Nichols, Indiana University (Center for Research on Concepts and Cogniton)

ISMIR 2010, Utrecht: Tutorial 2, August 9.

Supplementary material: http://aimusic.net/ismir2010tutorial/

1

Motivation: Representation

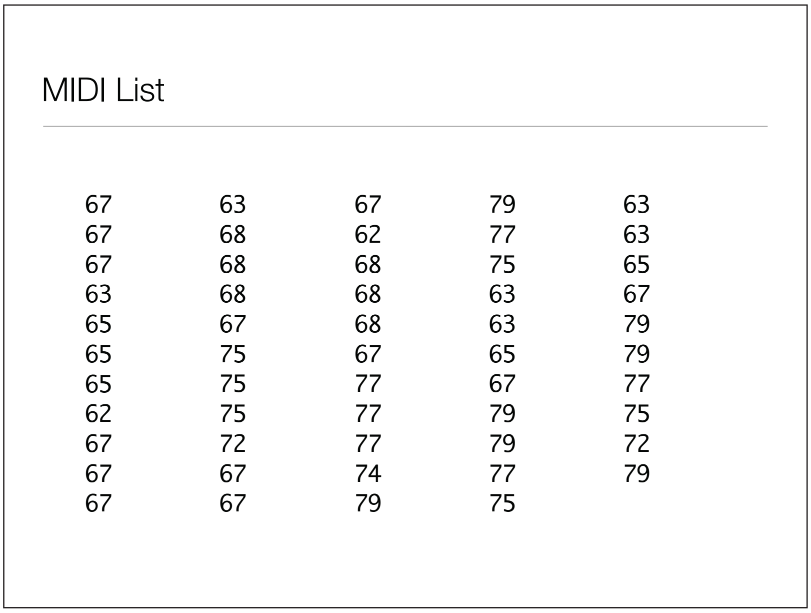
- Motif Search using Pitch:
  - Frequency List
  - Note List (MIDI)
  - Pitch Class List
  - Note Name List
  - Contour List
- Motif Search using Rhythm:
  - Onset time list
  - Duration list
  - Quantized duration
  - Quantized duration + metric position

2

Frequency List

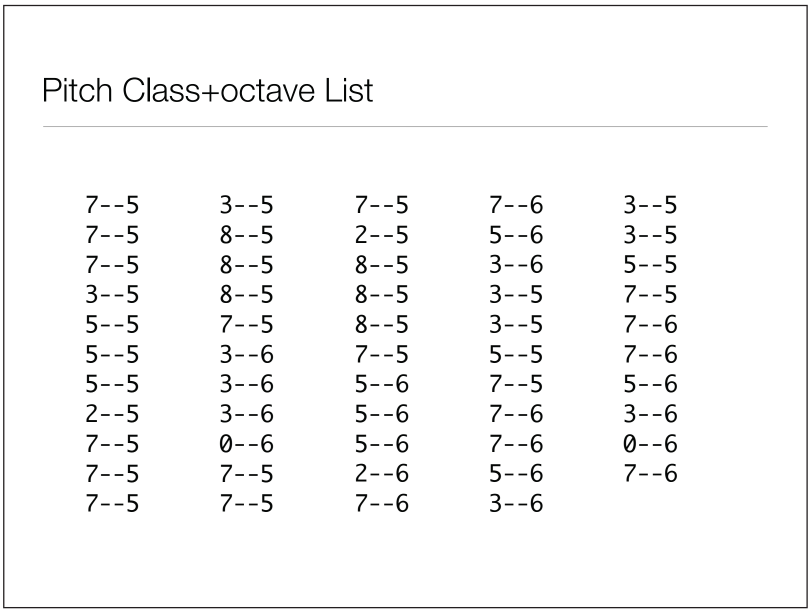
```
783.99    830.61    830.61    783.99    1046.50
783.99    830.61    783.99    1567.98   1567.98
783.99    783.99    1396.91   1567.98
622.25    1244.51   1396.91   1396.91
698.46    1244.51   1396.91   1244.51
698.46    1244.51   1174.66   622.25
698.46    1046.50   1567.98   622.25
587.33    783.99    1567.98   698.46
783.99    783.99    1396.91   783.99
783.99    783.99    1244.51   1567.98
783.99    587.33    622.25    1567.98
622.25    830.61    622.25    1396.91
830.61    830.61    698.46    1244.51
```

3



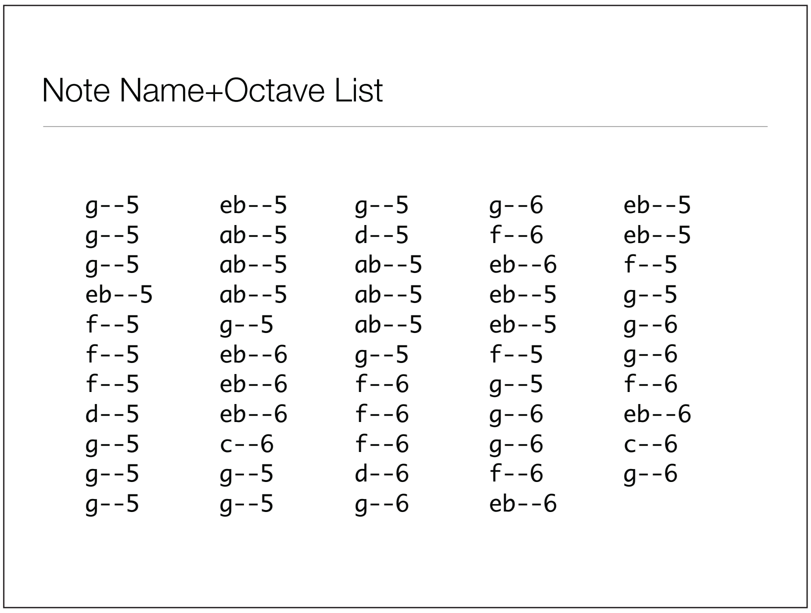
MIDI List

```
67
67
67
63
65
65
65
62
67
67
67
63
68
68
68
67
75
75
75
72
67
67
67
62
68
68
68
67
77
77
77
74
79
79
77
75
63
63
65
67
79
79
77
75
63
63
65
67
79
79
77
75
72
79
```



Pitch Class+octave List

```
7--5
7--5
7--5
3--5
5--5
5--5
5--5
2--5
7--5
7--5
7--5
3--5
8--5
8--5
8--5
7--5
3--6
3--6
3--6
0--6
7--5
7--5
7--5
2--5
8--5
8--5
8--5
7--5
5--6
5--6
5--6
2--6
7--6
7--6
5--6
3--6
3--5
3--5
5--5
7--5
7--6
7--6
5--6
3--6
3--5
3--5
5--5
7--5
7--6
7--6
5--6
3--6
0--6
7--6
```



Note Name+Octave List

```
g--5
g--5
g--5
eb--5
f--5
f--5
f--5
d--5
g--5
g--5
g--5
eb--5
ab--5
ab--5
ab--5
g--5
eb--6
eb--6
eb--6
c--6
g--5
g--5
g--5
d--5
ab--5
ab--5
ab--5
g--5
f--6
f--6
f--6
d--6
g--6
g--6
f--6
eb--6
eb--5
eb--5
f--5
g--5
g--6
g--6
f--6
eb--6
eb--5
eb--5
f--5
g--5
g--6
g--6
f--6
eb--6
c--6
g--6
```

Pitch Difference List

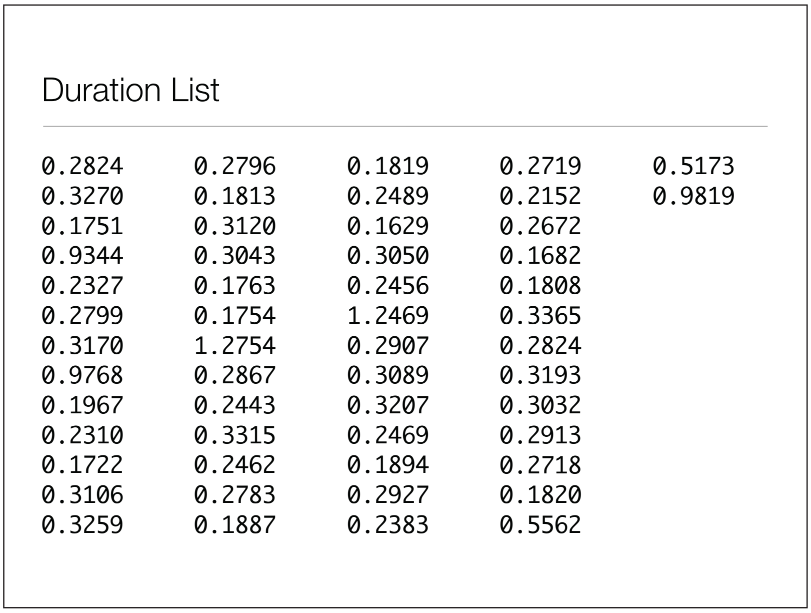
```
         -4       0        0        -12
0        5        -5       -2       0
0        0        6        -2       2
-4       0        0        -12      2
2        -1       0        0        12
0        8        -1       2        0
0        0        10       2        -2
-3       0        0        12       -2
5        -3       0        0        -3
0        -5       -3       -2       7
0        0        5        -2
```

Contour List

```
         D        -        -        D
-        U        D        D        -
-        -        U        D        U
D        -        -        D        U
U        D        -        -        U
-        U        D        U        -
-        -        U        U        D
D        -        -        U        D
U        D        -        -        D
-        D        D        D        U
-        -        U        D
```

Attack Time List

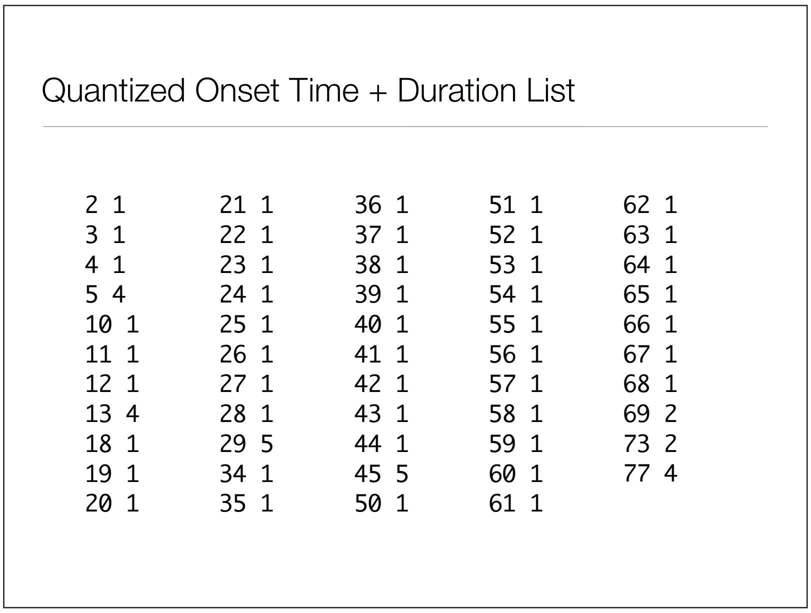
```
0.1688   5.0495   8.6805   12.5554  15.1822
0.4426   5.3014   8.9741   12.8144  15.4911
0.7097   5.5042   9.2092   13.0445  15.6690
1.0207   5.8264   9.5867   13.1750  16.0279
2.2210   5.9320   9.8003   13.5514  16.3091
2.5609   6.2199   10.0706  13.8179  16.5502
2.7823   6.5455   10.2715  14.0378  16.7820
3.0579   6.8126   10.5120  14.2627  16.9670
4.1745   6.9745   10.7509  14.5685  17.9614
4.4379   8.2133   10.9269  14.7315  19.0543
4.7373   8.4183   12.3120  15.0575
```



Duration List

```
0.2824    0.2796    0.1819    0.2719    0.5173
0.3270    0.1813    0.2489    0.2152    0.9819
0.1751    0.3120    0.1629    0.2672
0.9344    0.3043    0.3050    0.1682
0.2327    0.1763    0.2456    0.1808
0.2799    0.1754    1.2469    0.3365
0.3170    1.2754    0.2907    0.2824
0.9768    0.2867    0.3089    0.3193
0.1967    0.2443    0.3207    0.3032
0.2310    0.3315    0.2469    0.2913
0.1722    0.2462    0.1894    0.2718
0.3106    0.2783    0.2927    0.1820
0.3259    0.1887    0.2383    0.5562
```

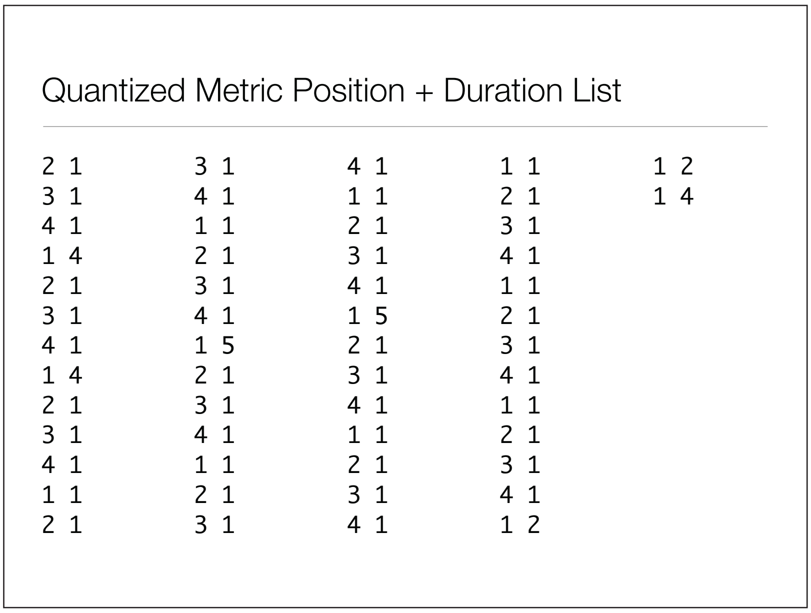
10



Quantized Onset Time + Duration List

```
   2 1      21 1      36 1      51 1      62 1
   3 1      22 1      37 1      52 1      63 1
   4 1      23 1      38 1      53 1      64 1
   5 4      24 1      39 1      54 1      65 1
   10 1     25 1      40 1      55 1      66 1
   11 1     26 1      41 1      56 1      67 1
   12 1     27 1      42 1      57 1      68 1
   13 4     28 1      43 1      58 1      69 2
   18 1     29 5      44 1      59 1      73 2
   19 1     34 1      45 5      60 1      77 4
   20 1     35 1      50 1      61 1
```

11



Quantized Metric Position + Duration List

```
2 1       3 1       4 1       1 1       1 2
3 1       4 1       1 1       2 1       1 4
4 1       1 1       2 1       3 1
1 4       2 1       3 1       4 1
2 1       3 1       4 1       1 1
3 1       4 1       1 5       2 1
4 1       1 5       2 1       3 1
1 4       2 1       3 1       4 1
2 1       3 1       4 1       1 1
3 1       4 1       1 1       2 1
4 1       1 1       2 1       3 1
1 1       2 1       3 1       4 1
2 1       3 1       4 1       1 2
```

12

Metric Position + Duration + Grouping

```
2 1         2 1         1 1         4 1         3 1
3 1         3 1                     1 1         4 1
4 1         4 1         2 1                     1 1
1 4         1 1         3 1         2 1
                        4 1         3 1         2 1
2 1         2 1         1 1         4 1         3 1
3 1         3 1                     1 1         4 1
4 1         4 1         2 1                     1 2
1 4         1 5         3 1         2 1
                        4 1         3 1         1 2
2 1         2 1         1 5         4 1
3 1         3 1                     1 1         1 4
4 1         4 1         2 1
1 1                     3 1         2 1
```

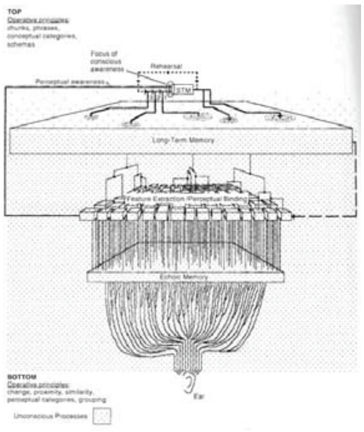
Overview

- Principles of Music Cognition
- Models of Grouping and Segmentation
- Models of Motif Recognition
- Models of Music Cognition/Listening

I. Principles of Music Cognition

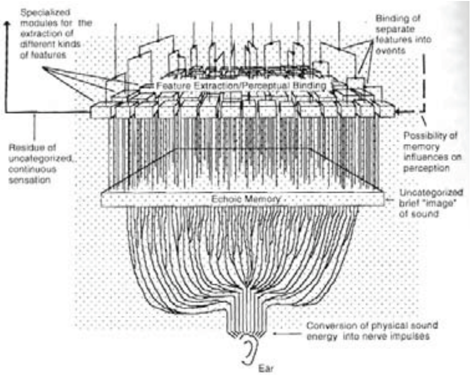
- Music and Memory
- Gestalt Perception
- Encoding of Rhythm and Pitch
- Expectation

Music and Memory (Snyder 2000)



16

Music and Memory



17

Music and Memory

	Events per second	Seconds per event
Event Fusion: early processing	16,384	1/16,384
	32	1/32
Melodic/Rhythmic Grouping: short term memory	16	1/16
	1	1
	1/8	8
Form: long term memory	1/16	16
	1/4,096	1hr+

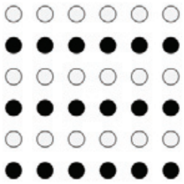
18

Gestalt Principles

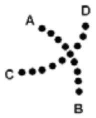
Gestalt Principles

- Gestalt theory for visual perception
  - Proximity
  - Similarity
  - Good Continuation
  - Multistability

Gestalt Principles

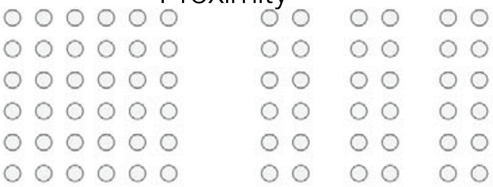


Similarity

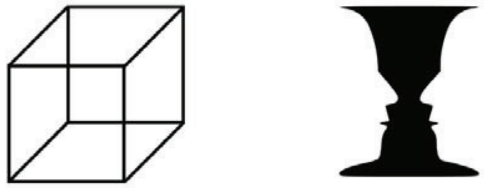


Good Continuation

Proximity



Gestalt Principles



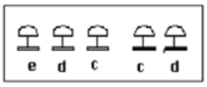
Multistability

Encoding of Rhythm & Pitch

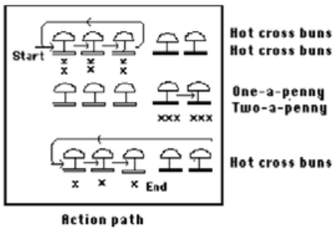
- Bamberger: Childhood development and music cognition
- Povel and Essens: Rhythmic representation
- Deutsch and Feroe: Pitch Alphabets

Bamberger

- Montessori bells

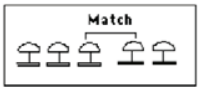


- Hot Cross Buns

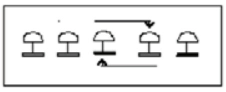


Bamberger

- Prompt 1: “Are there two bells that sound the same?”



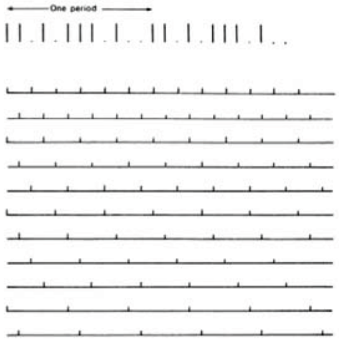
- Prompt 2: “Can you play the piece in a different way, knowing that these two match?”



25

Encoding of Rhythm & Pitch

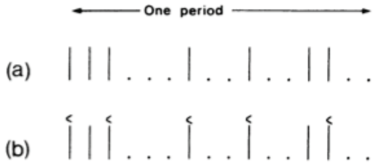
- Povel and Essens (1985): Encoding Rhythm
  - Rhythmic patterns induce an internal clock
  - The clock is hierarchical -- beats are subdivided
  - Rhythmic patterns can be encoded with respect to the clock



26

Encoding of Rhythm

- Perceived accents:
  - Isolated
  - Initial (Primacy)
  - Final (Recency)



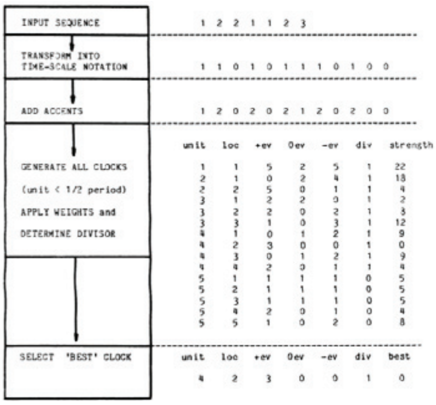
27

Encoding of Rhythm

- Povel and Essens generate a list of repeating rhythmic sequences.
- Then, a collection of 2-level hierarchical clocks is generated
- Each clock is scored for each pattern, based on how the accents of the rhythm match up with the subdivisions of the clock.
- Two experiments:
  - People more accurately reproduce patterns suggested by clocks
  - People do even better when the best clock is presented in the stimulus

Encoding of Rhythm

- +ev: count how many times an accented event lines up with a clock event
- 0ev: count # of matches of clock events with rests
- -ev: count # of matches of clock events with unaccented events
- div: does the clock length evenly divide the period?



1 2 2 1 1 2 3

1 1 0 1 0 1 1 1 0 1 0 0

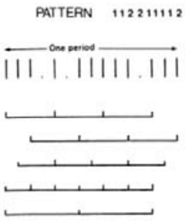
1 2 0 2 0 2 1 2 0 2 0 0

unit	loc	+ev	0ev	-ev	div	strength
1	1	5	2	5	1	22
2	1	0	2	4	1	18
2	2	5	0	1	1	4
3	1	2	2	0	1	2
3	2	2	0	2	1	3
3	3	1	0	3	1	12
4	1	0	1	2	1	9
4	2	3	0	0	1	0
4	3	0	1	2	1	9
4	4	2	0	1	1	4
5	1	1	1	1	0	5
5	2	1	1	1	0	5
5	3	1	1	1	0	5
5	4	2	0	1	0	4
5	5	1	0	2	0	8

unit	loc	+ev	0ev	-ev	div	best
4	2	3	0	0	1	0

Encoding of Rhythm

- Hypothesis: the an internal clock allows for an efficient encoding of a rhythmic pattern.
- Tested by presenting identical patterns with two different clocks -- subjects indicated that the patterns were simpler for the better-matching clock, and usually didn't indicate that the patterns were identical.



Clock	Unit	Loc.	Code
1	4	1	1-1-2 , 2-1-1 , 1-1-2
2	4	3	S_2,S_4 , 2-1-1
3	3	2	1-2 , 2-1,S_3 , 2-1
4	2	1	S_2 , E , E , S_3,S_2 ,E
5	6	1	1-1-2-2 , 1-1-1-1-2

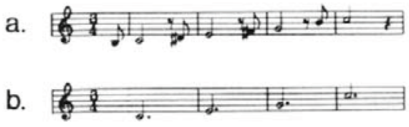
Encoding of Pitch

- Deutsch & Feroe (1981)
- according to Larson (2004):
  - [D&F] “describes musical passages in terms of alphabets (e.g., the chromatic scale, the major scale, and specific chords) and operations (e.g., repetition, or motion to the next higher or lower member) that create motions through those alphabets.”

Encoding of Pitch

- Alphabet selection: pedal mechanism of a harp

Encoding of Pitch

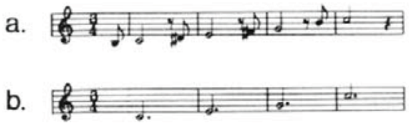


- A melody with two different hierarchical levels
- Each level is encoded using a different alphabet
  - a) Chromatic alphabet
  - b) C major triad alphabet

Encoding of Pitch

- Key principles of this pitch encoding:
  - Based on “overlearned alphabets”:
    - Chromatic, major/minor scales, major/minor triads, 7th chords, pentatonic, etc
  - Efficient encoding, reduces # and size of structures maintained by listener
  - Multiple encodings possible for one melody

Encoding of Pitch



- A = {(*, 3n); C_{tr}}   <-- line b)
- B = {(p, *); Cr}   <-- chromatic neighbor
- S = {A[pr]B; c}   <-- line a) (pr=”prime”)

Expectation

- Huron: ITPRA Theory
  - Imagination Response
  - Tension Response
  - Prediction Response
  - Reaction Response
  - Appraisal Response

Expectation: ITPRA

- Imagination Response
  - Imagining an outcome in advance allows us to feel some amount of pleasure or displeasure as we anticipate the future.
  - This helps motivate current behavior, to plan for future rewards
  - Think of feelings as an emotional landscape: the imagination reponse allows us to descend from a local maximum and journey to a better place.

37

Expectation: ITPRA

- Tension Response
  - Just before an event happens, we may prepare physically and mentally for the event; arousal and attention are increased to respond.

38

Expectation: ITPRA

- Prediction Response
  - One an event occurs, we have a positive emotional response based on correct prediction, and a negative response for an incorrect prediction.
  - This happens even if the expected outcome is bad. We are rewarded for accurate prediction, regardless of the outcome.

39

Expectation: ITPRA

- Reaction Response
  - Reaction to the actual outcome. Example: reflex response to touching a hot stove.
  - This is a fast response (not necessarily a reflex, but still a snap judgment).

40

Expectation: ITPRA

- Appraisal Response
  - Complex assessment of the final outcome, based on conscious thought.
  - Acts as behavioral reinforcement.

41

Expectation: ITPRA

- ITPRA theory is used throughout Huron’s book to discuss expectation in music
- Examples: syncopation, cadence, meter, tonality, and climax exploit the ITPRA process, resulting in various types of musical affect

42

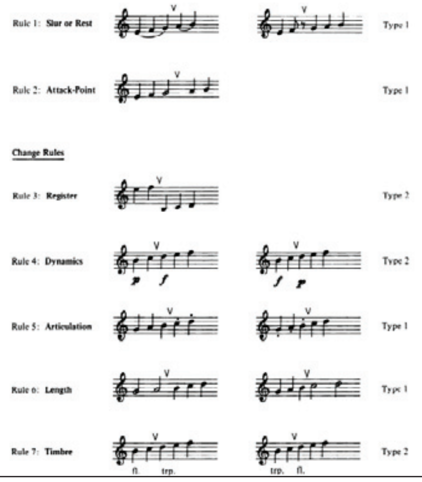
II. Models of Grouping & Segmentation

- GTTM
- Deliege
- Temperley
- Ferrand/Nelson/Wiggins

Lerdahl and Jackendoff: GTTM

- Grouping Preference Rules
- Inspired by Gestalt proximity and similarity
- Main idea for grouping:
    - Find discontinuities along some dimension of music (pitch, rhythm, etc)
    - A segmentation is implied at these discontinuities, with strength relative to the strength of the discontinuity

GTTM Grouping Preference Rules (from Deliège)



Deliège

- Analyzed grouping rules in GTTM with two objectives:
  1. Confirm validity of the rules
  2. Determine relative importance of rules
- Important grouping factors
  - Group boundaries after rests/slurs
  - At long intervals between attacks
  - At changes in articulation, dynamics, note length, register

Deliège

- Factors categorized in Gestalt terms:
  - Gestalt Proximity (in time)
    - Rests
    - Attack Points
  - Gestalt Similarity
    - Changes based on dynamics, note length, register, articulation

Deliège

- Relative importance of GTTM grouping preference rules:
  - Change of timbre (GPR 7) was most important
  - Followed by: register (GPR 3) and attack point (GPR 2)
-

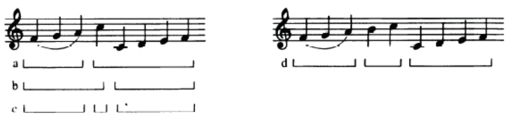


Fig. 4. Conflict between R. 1 (Slur and rest) and R. 3 (Change of register). (a) Preference for Slur and rest. (b) Preference for Change of register. (c) Not a preferred segmentation because of a group with one sound. (d) Conflict removed by adding one sound.

Temperley

- Preference rules for segmentation (Phrase structure preference rules, PSPRs)
  - PSPR 1: Gap rule
  - PSPR 2: Phrase length rule
  - PSPR 3: Metrical parallelism rule
- Tested on 65 songs from Essen. 75.5% success rate.

49

Ferrand/Nelson/Wiggins

- Melodic segmentation model based on melodic density (MDSM). Models short-term memory, time, Gestalt proximity
- Model:
  - Compute melodic density
    - Based on computing pitch intervals between all notes in short-term memory (uses a sliding window of time)
    - Recent pitches and proximate pitches are weighed more heavily
    - Intervals common in major/minor scales are weighed more heavily
  - Find local minima in density, and identify these points as boundaries
- Compared to Cambouropoulos' Local Boundary Detection Model: MDSM is more conservative.

50

III. Models of Motif Recognition

- Knopke (see above)
- Lartillot
- Nichols

51

Lartillot: Motif Recognition

- Automatic motive extraction in melodies
- Based on many different features of the melodic surface
- General idea: look for patterns that repeat, and label them **motives**.
  - **themes** are the most important motives of a piece
  - **signatures** (Cope) are stylistic repeated features

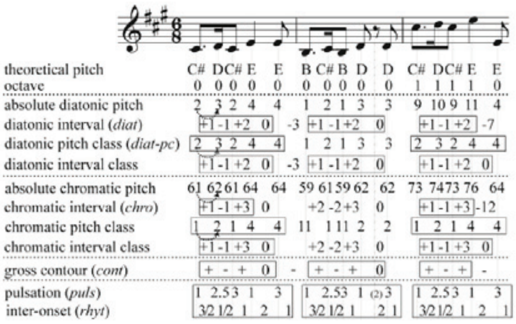
Lartillot: Motif Recognition

- Difficulties with motif extrction
  - Ensure musical interest
    - (one solution/new problem: ensure perceptual relevance)
  - Combinatoric explosion of possible matches across different parameters

Lartillot: Motif Recognition

- Solution here:
  - Match “heterogeneous motives”
  - Perform exact matching, but only on certain features in the feature set
  - Force the beginning parts of new possible motif instances to use an exact match (inspired by LTM search), but allow generalization to less features in the match for the ends of motifs

Lartillot: Motif Recognition



theoretical pitch	C# D C# E E B C# B D D C# D C# E E
octave	0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
absolute diatonic pitch	2 3 2 4 4 1 2 1 3 3 9 10 9 11 4
diatonic interval (*diat*)	+1 -1 +2 0 -3 +1 -1 +2 0 +1 -1 +2 -7
diatonic pitch class (*diat-pc*)	2 3 2 4 4 1 2 1 3 3 2 3 2 4 4
diatonic interval class	+1 -1 +2 0 -3 +1 -1 +2 0 +1 -1 +2 0
absolute chromatic pitch	61 62 61 64 64 59 61 59 62 62 73 74 73 76 64
chromatic interval (*chro*)	+1 -1 +3 0 +2 -2 +3 0 +1 -1 +3 -12
chromatic pitch class	1 2 1 4 4 11 1 11 2 2 1 2 1 4 4
chromatic interval class	+1 -1 +3 0 +2 -2 +3 0 +1 -1 +3 0
gross contour (*cont*)	+ - + 0 - + - + 0 + - + -
pulsation (*puls*)	1 2.5 3 1 3 1 2.5 3 1 (2) 3 1 2.5 3 1 3
inter-onset (*rhyt*)	3/2 1/2 1 2 1 3/2 1/2 1 2 1 3/2 1/2 1 2 1

Figure 1.

Descriptions of a monody. Repeated sequences of values, forming patterns, are enclosed in boxes.

IV. Models of Music Cognition

- Narmour (I-R, Tonal Pitch Space)
- Margulis (Dynamic melodic expectation)
- Larson (Seek Well)
- Nichols

Narmour’s I-R Theory

- Two universal formal hypotheses (in terms of form, intervals, or pitches):
  - A + A -> A
  - A + B -> C

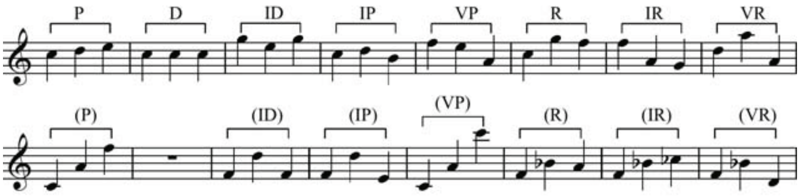
Narmour's I-R Theory

- Five melodic archetypes:
  - Process [P], Duplication [D] (A+A, no closure)
  - Reversal [R] (A+B, with closure)
  - Registral return [aba]
  - Dyad (two implicative terms, no realization)
  - Monad (One element, no implication)

Narmour's I-R Theory

- P, D, and R symbols: direction and interval sizes are acting together
- Five more symbols for other cases:
  - IP: intervallic process
  - VP: registral process
  - IP: intervallic reversal
  - VR: registral reversal
  - ID: intervallic duplication

Narmour's I-R Theory



Narmour's I-R Theory

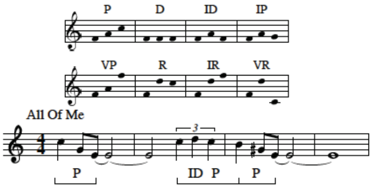
- Other aspects of theory
- Top-Down, Bottom-up
- Schellenberg's simplified model
- Related work (focused on pitch-triples)

Applications of I-R

- Melody Retrieval
- Expressive Performance
- Performer Identification in Celtic Violin (Ramirez et al)

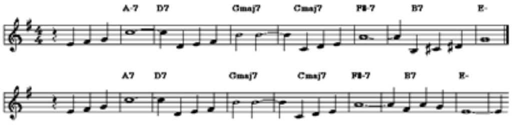
Melody Retrieval

- Grachten, Arcos, Mántaras, MIREX '05
- Automatically computes the amount of closure at each point in a melody based on metrical/rhythmic criteria
- Annotates melody with I-R symbols
- Edit distance defined for I-R symbols
- Weights chosen via evolutionary method



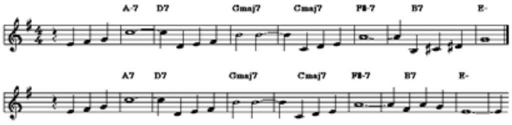
Expressive Performance

- SaxEx (Mántaras & Arcos)
  - ex: http://www.iiia.csic.es/~arcos/noos/Demos/Example.html



Expressive Performance

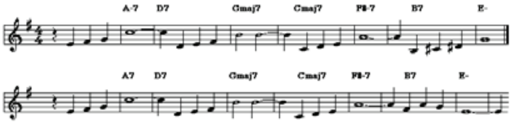
- SaxEx (Mántaras & Arcos)
  - Case-Based Reasoning (CBR) system for expressive performance
  - Retreival is performed by combining an I-R analysis with GTTM metrical importance criteria. http://www.iiia.csic.es/~arcos/noos/Demos/Example.html



Inexpressive

Expressive Performance

- SaxEx (Mántaras & Arcos)
  - Case-Based Reasoning (CBR) system for expressive performance
  - Retreival is performed by combining an I-R analysis with GTTM metrical importance criteria. http://www.iiia.csic.es/~arcos/noos/Demos/Example.html



Expressive

Performer Identification in Celtic Violin (Ramirez et al, 2008)

- Task: identifying performers based on a performance
- Relevant input data: note timing and amplitude deviations
- Scores analyzed via I-R symbols -- along with information about the pitch, duration, and context, I-R symbols are included in a tuple fed to a classifier.

67

Margulis's Expectation Model

- Combination of several models
  - Narmour's I-R model
  - Lerhadl's Tonal Pitch Space model
  - GTTM preference rules

68

Margulis's Expectation Model

- Generates graphs of 3 types of tension
  - **Surprise-tension** (amount of unexpectedness, intensity, dynamism
  - **Denial-tension** (A...B... ***F#)***, not C ("will, intention, determinedness"
  - **Expectation-tension** (A..B...?) C expected strongly (Forward-looking calculation of "yearning")

69

Surprise-tension

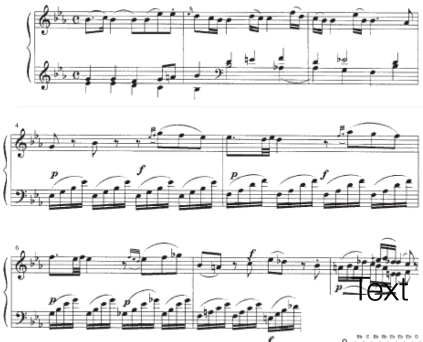


Fig. 9. Mozart, Piano Sonata K. 282, measures 1–8.

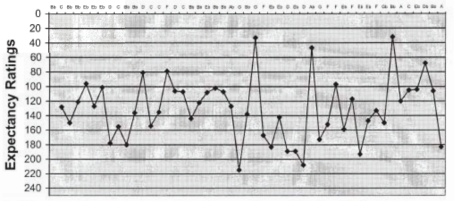


Fig. 10. Predicted overall surprise-tension across the course of the melody of measures 1–8 of Mozart K. 282, where the tempo is quarter = 48. The *x* axis represents pitch events, and the *y* axis represents expectancy ratings, listed from high to low so that the inversely proportional surprise-tension can be seen to increase with graph height.

70

Denial-tension

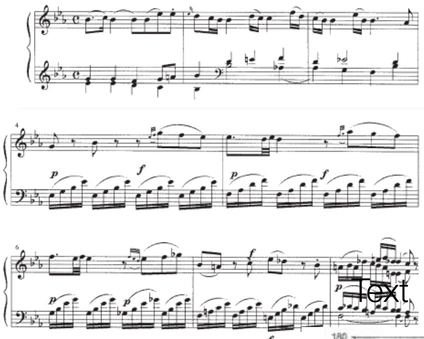


Fig. 9. Mozart, Piano Sonata K. 282, measures 1–8.

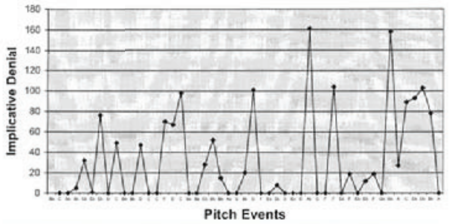


Fig. 12. Predicted overall denial-tension across the course of the melody of measures 1–8 of Mozart K. 282, where the tempo is quarter = 48.

71

Expectancy-tension

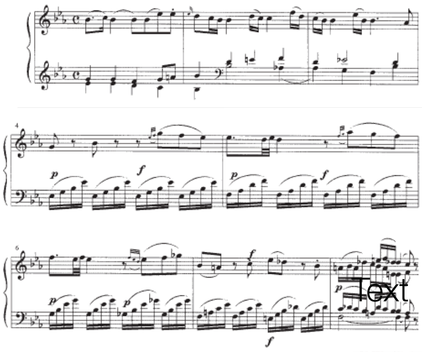


Fig. 9. Mozart, Piano Sonata K. 282, measures 1–8.

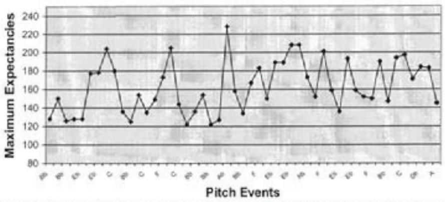


Fig. 13. Predicted overall expectancy-tension across the course of the melody of measures 1–8 of Mozart K. 282, where the tempo is quarter = 48.

72

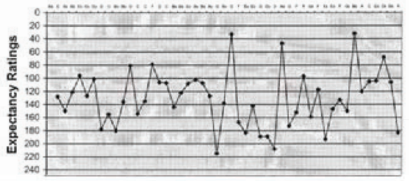


Fig. 10. Predicted overall surprise-tension across the course of the melody of measures 1–8 of Mozart K. 282, where the tempo is quarter = 48. The x axis represents pitch events, and the y axis represents expectancy ratings, listed from high to low so that the inversely proportional surprise-tension can be seen to increase with graph height.

Surprise-tension

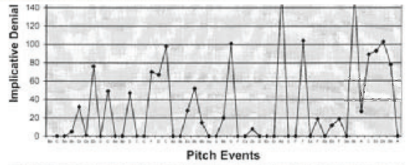


Fig. 12. Predicted overall denial-tension across the course of the melody of measures 1–8 of Mozart K. 282, where the tempo is quarter = 48.

Denial-tension

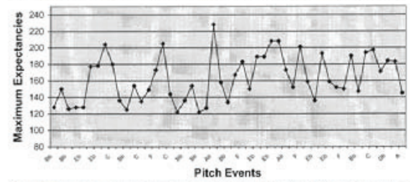
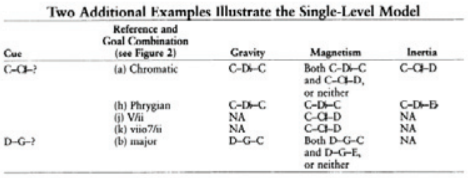


Fig. 13. Predicted overall expectancy-tension across the course of the melody of measures 1–8 of Mozart K. 282, where the tempo is quarter = 48.

Expectancy-tension

Larson's Model of Musical Forces

- 3 Musical forces for melodic expectation:
  - Inertia
  - Magnetism
  - Gravity

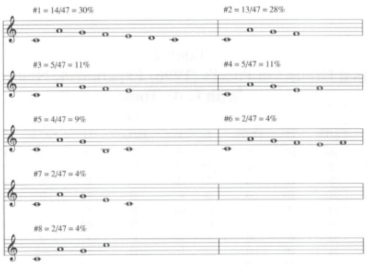


Two Additional Examples Illustrate the Single-Level Model

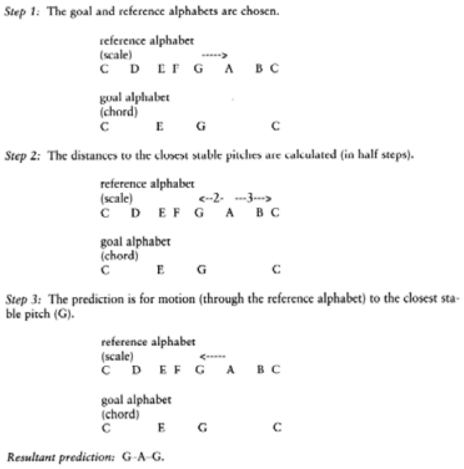
Cue	Reference and Goal Combination (see Figure 2)	Gravity	Magnetism	Inertia
C–C♯–?	(a) Chromatic	C–D♭–C	Both C–D♭–C and C–C♯–D, or neither	C–C♯–D
	(h) Phrygian	C–D♭–C	C–D♭–C	C–D♭–E♭
	(j) V/ii	NA	C–C♯–D	NA
	(k) viio7/ii	NA	C–C♯–D	NA
D–G–?	(b) major	D–G–C	Both D–G–C and D–G–E, or neither	NA

Larson's SeekWell Model

- Melodic expectation model based on:
  - 3 musical forces
  - A pitch alphabet (simple model)
  - Hierarchical alphabets (multi-level model)



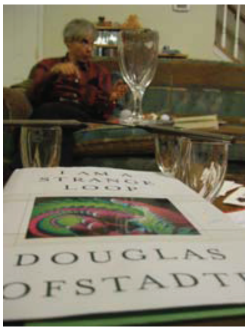
Larson’s SeekWell Model



```
Step 1:  The goal and reference alphabets are chosen.

                    reference alphabet
                    (scale)              ----->
                    C    D    E F   G     A     B C

                    goal alphabet
                    (chord)
                    C         E     G           C

Step 2:  The distances to the closest stable pitches are calculated (in half steps).

                    reference alphabet
                    (scale)             <--2-  ---3--->
                    C    D    E F   G     A     B C

                    goal alphabet
                    (chord)
                    C         E     G           C

Step 3:  The prediction is for motion (through the reference alphabet) to the closest sta-
ble pitch (G).

                    reference alphabet
                    (scale)             <-----
                    C    D    E F   G     A     B C

                    goal alphabet
                    (chord)
                    C         E     G           C

Resultant prediction:  G-A-G.
```

Nichols and Hofstadter: Musicat

- Musicat: Inspired by
  - Larson’s SeekWell
  - Mitchell’s Copycat

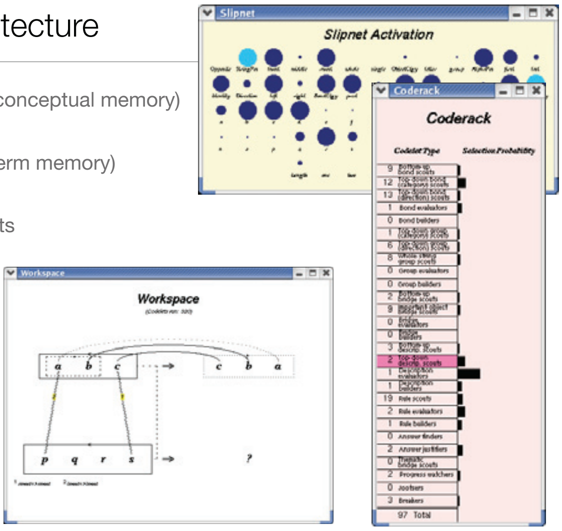


Copycat

- A B C -> A B D
  Q R S -> ?
- A B C -> A B D
  X Y Z -> ?

Copycat architecture

- Slipnet (Long term conceptual memory)
- Workspace (Short term memory)
- Coderack & Codelets
- Temperature

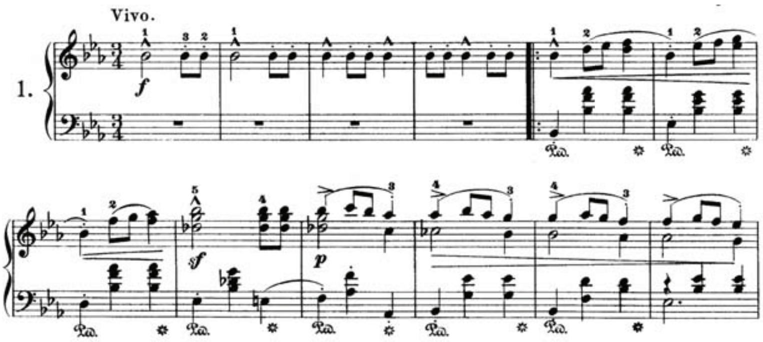


Coderack

Codelet Type		Selection Probability
9	Bottom-up bond scouts	
12	Top-down bond (category) scouts	
13	Top-down bond (direction) scouts	
1	Bond evaluators	
0	Bond builders	
1	Top-down group (category) scouts	
6	Top-down group (direction) scouts	
8	Whole string group scouts	
0	Group evaluators	
0	Group builders	
2	Bottom-up bridge scouts	
9	Important object bridge scouts	
0	Bridge evaluators	
0	Bridge builders	
3	Bottom-up descrip. scouts	
2	Top-down descrip. scouts	
1	Description evaluators	
1	Description builders	
19	Rule scouts	
2	Rule evaluators	
1	Rule builders	
0	Answer finders	
2	Answer justifiers	
0	Thematic bridge scouts	
2	Progress watchers	
0	Jootsers	
3	Breakers	
97	Total	

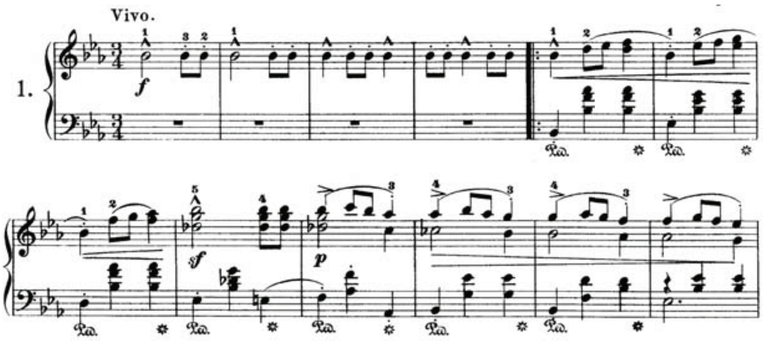
Musicat

- Demo



Musicat

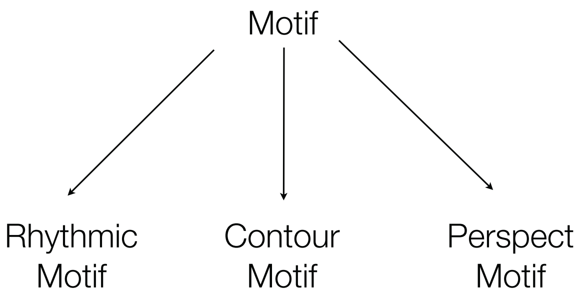
- Demo



Musicat

- Simulates listening to music in time
- Input: symbolic notes
- Output: grouping structures and expectations
- Codelets: simulates Gestalt perception, preference rules for grouping
- Conceptual Memory: active LTM affecting perception
- Analogy as the core

Musicat: Representation


Motif

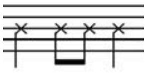
Rhythmic Motif

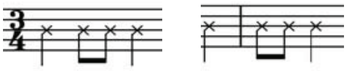
Contour Motif

Perspect Motif

Musicat: Representation

- Rhythmic Motif
    - Note durations (quarter, half, etc.)
    - Optional: metric position of attacks



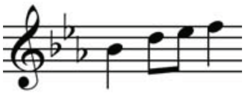


Musicat: Representation

- Contour Motif
  - Up, Down, or Sideways motion
  - Optional:
    - Specified alphabet (Chromatic, diatonic, major triad, etc)
    - Distance in alphabet

Musicat: Representation

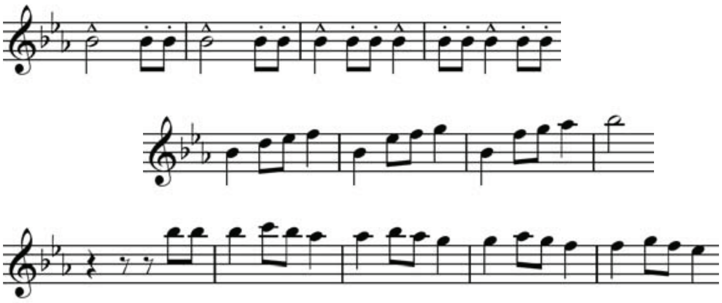
- Perspect: “PERceived aSPECT” (Ockenfeld)
- Perspect motif: Each note has a perspect list
- Example:



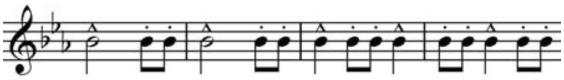
  - Bb: (downbeat=80, dominant=80, group start=70, Bb=60)
  - D: (group start=50, leading-tone=20)
  - Eb: (passing tone=80)

Musicat: Representation

- Motif Generalization



Musicat: Representation



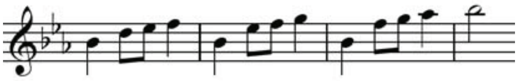
a) Half(1), eighth(3), eighth(3.5)

b) First attempt:
Quarter (1), eighth(2), eighth(2.5)

c) Generalization:
Quarter, eighth, eighth

88

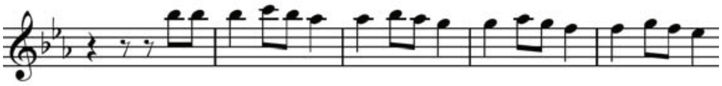
Musicat: Representation



R: Quarter(1), eighth(2), eighth(2.5), quarter(3)
C: Up, up, up
P: (Beat 1: Bb=80, dominant=80)

89

Musicat: Representation



R: Quarter(1), eighth(2), eighth(2.5), quarter(3)
C: Up, down, down
P: (Beat 1: Suspension=70, ...)
(Beat 3: Chord tone=80, ...)

90

Musicat: Codelets

- Grouping Scouts
  - Pitch Gap Scout
  - Rhythmic Gap Scout
  - Downbeat
  - Repetition Scout
  - Linear Connection Scout

91

Musicat: Codelets

- Groupers:
  - Scout-based groupers
  - Motif groupers
  - Meta-groupers
  - Group extenders
- Motif codelets (top-down)
- Analogy-mapping

92

Applications

- Music Cognition as a tool for improved representational schemes

93

Applications

- Creativity
  - Creative Listening
    - Listening for Retrieval
    - Modeling Listening = modeling emotional response
  - Creative Performance
  - Creative Composition

Acknowledgements

- The authors would like to express their thanks for the generous support provided by their respective institutions:
  - the BBC
  - Douglas Hofstadter's Center for Research on Concepts and Cognition at Indiana University.
  - NSF CreativeIT grant #0738384.

- Supplementary info: http://aimusic.net/ismir2010tutorial/