

Lecture
Music Processing

Audio Retrieval

Meinard Müller

International Audio Laboratories Erlangen
meinard.mueller@audiolabs-erlangen.de

Music Retrieval

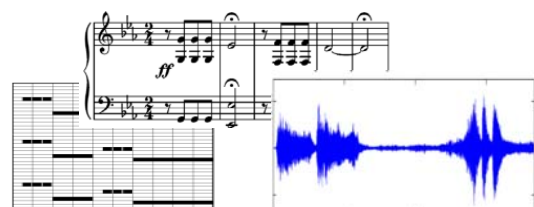
- Textual metadata
 - Traditional retrieval
 - Searching for artist, title, ...
- Rich and expressive metadata
 - Generated by experts
 - Crowd tagging, social networks
- Content-based retrieval
 - Automatic generation of tags
 - Query-by-example

Google™

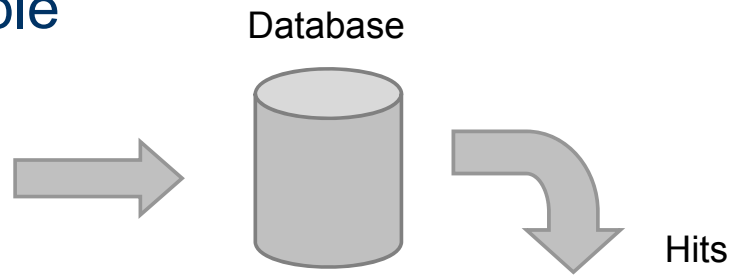
Beethoven

beethoven
beethoven **biography**
beethoven **movie**
beethoven **music**
beethoven's **5th**

acoustic alternative ambient baroque beautiful beethoven blues
britpop celtic chillout classica **classical** classic rock
classical music classical period
classique composer **composers** contemporary classical easy
listening electronic favourite folk funk genius german
germany hard rock indie **instrumental** jazz klassik latin
love ludwig van beethoven mellow metal opera orchestra
orchestral piano pop power metal progressive progressive rock
psychedelic punk rock **romantic** romantic classical romantic
period romanticism sexy singer-songwriter ska soul stoner rock
symphonic symphony



Query-by-Example



Retrieval tasks:

- Audio identification
- Audio matching
- Version identification
- Category-based music retrieval

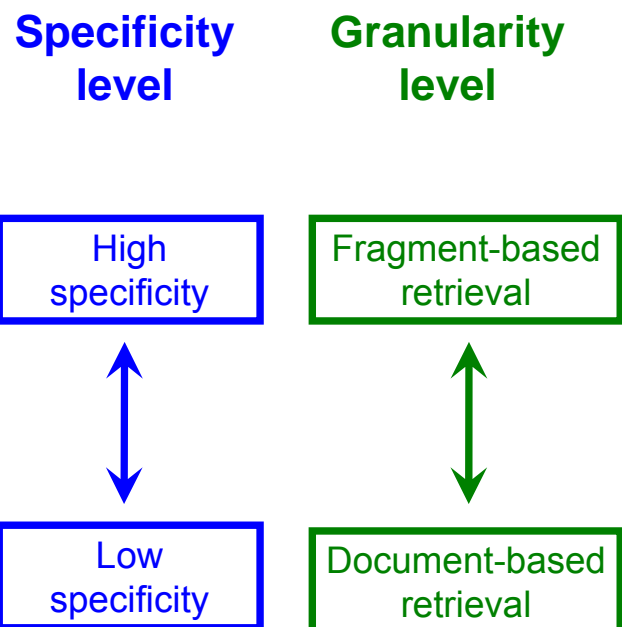
- Bernstein (1962)
- Beethoven, Symphony No. 5
- Beethoven, Symphony No. 5:
 - Bernstein (1962)
 - Karajan (1982)
 - Gould (1992)
- Beethoven, Symphony No. 9
- Beethoven, Symphony No. 3
- Haydn Symphony No. 94

Query-by-Example

Taxonomy

Retrieval tasks:

- Audio identification
- Audio matching
- Version identification
- Category-based music retrieval

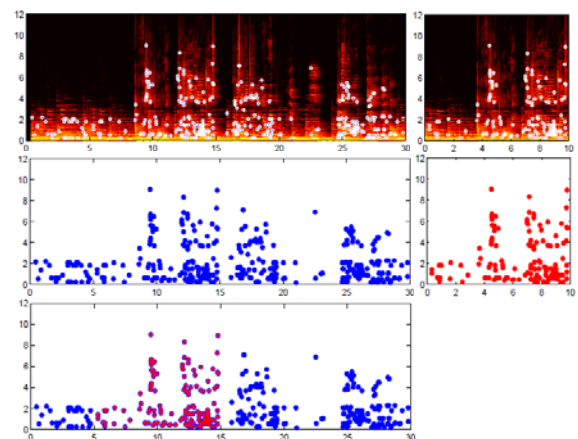


Overview (Audio Retrieval)

- Audio identification (audio fingerprinting)
- Audio matching
- Cover song identification

Overview (Audio Retrieval)

- Audio identification (audio fingerprinting)
- Audio matching
- Cover song identification



Audio Identification

- Database:** Huge collection consisting of all audio recordings (feature representations) to be potentially identified.
- Goal:** Given a short **query audio fragment**, identify the original audio recording the query is taken from.
- Notes:**
- Instance of fragment-based retrieval
 - High specificity
 - Not the piece of music is identified but a specific rendition of the piece

Application Scenario

- User hears music playing in the environment
- User records music fragment (5-15 seconds) with mobile phone
- Audio fingerprints are extracted from the recording and sent to an audio identification service
- Service identifies audio recording based on fingerprints
- Service sends back metadata (track title, artist) to user

Audio Fingerprints

An **audio fingerprint** is a content-based compact signature that summarizes some specific audio content.

Requirements:

- Discriminative power
- Invariance to distortions
- Compactness
- Computational simplicity

Audio Fingerprints

An **audio fingerprint** is a content-based compact signature that summarizes a piece of audio content

Requirements:

- **Discriminative power**
- Invariance to distortions
- Compactness
- Computational simplicity

- *Ability to accurately identify an item within a huge number of other items (informative, characteristic)*
- *Low probability of false positives*
- *Recorded query excerpt only a few seconds*
- *Large audio collection on the server side (millions of songs)*

Audio Fingerprints

An **audio fingerprint** is a content-based compact signature that summarizes a piece of audio content

Requirements:

- Discriminative power
- **Invariance to distortions**
- Compactness
- Computational simplicity

- *Recorded query may be distorted and superimposed with other audio sources*
- *Background noise*
- *Pitching (audio played faster or slower)*
- *Equalization*
- *Compression artifacts*
- *Cropping, framing*
- *...*

Audio Fingerprints

An **audio fingerprint** is a content-based compact signature that summarizes a piece of audio content

Requirements:

- Discriminative power
- Invariance to distortions
- **Compactness**
- Computational simplicity

- *Reduction of complex multimedia objects*
- *Reduction of dimensionality*
- *Making indexing feasible*
- *Allowing for fast search*

Audio Fingerprints

An **audio fingerprint** is a content-based compact signature that summarizes a piece of audio content

Requirements:

- Discriminative power
- Invariance to distortions
- Compactness
- **Computational simplicity**

- *Computational efficiency*
- *Extraction of fingerprint should be simple*
- *Size of fingerprints should be small*

Literature (Audio Identification)

- Allamanche et al. (AES 2001)
- Cano et al. (AES 2002)
- Haitsma/Kalker (ISMIR 2002)
- Kurth/Clausen/Ribbrock (AES 2002)
- Wang (ISMIR 2003)
- ⋮
- Dupraz/Richard (ICASSP 2010)
- Ramona/Peeters (ICASSP 2011)

PHILIPS

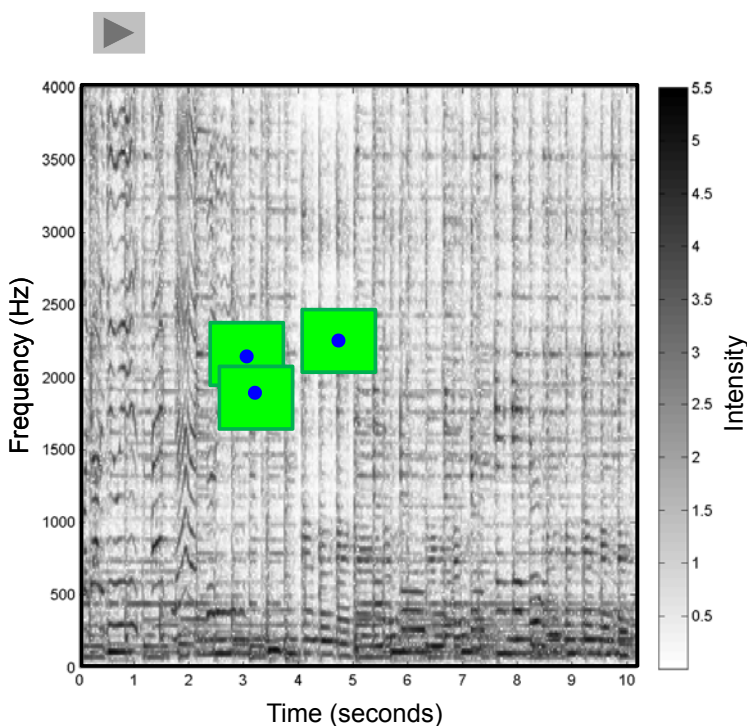


Literature (Audio Identification)

- Allamanche et al. (AES 2001)
- Cano et al. (AES 2002)
- Haitisma/Kalker (ISMIR 2002)
- Kurth/Clausen/Ribbrock (AES 2002)
- Wang (ISMIR 2003)
- ⋮
- Dupraz/Richard (ICASSP 2010)
- Ramona/Peeters (ICASSP 2011)



Fingerprints (Shazam)

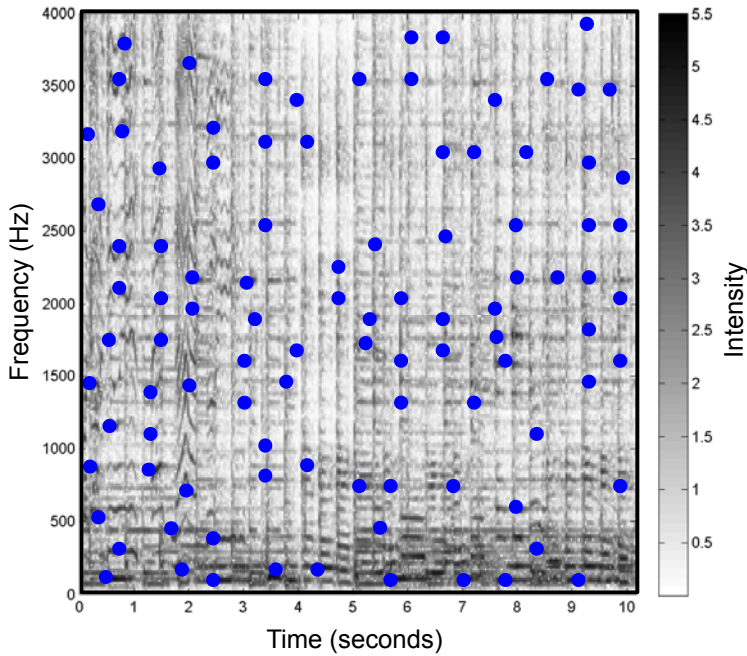


Steps:

1. Spectrogram
2. Peaks
(local maxima)

- *Efficiently computable*
- *Standard transform*
- *Robust*

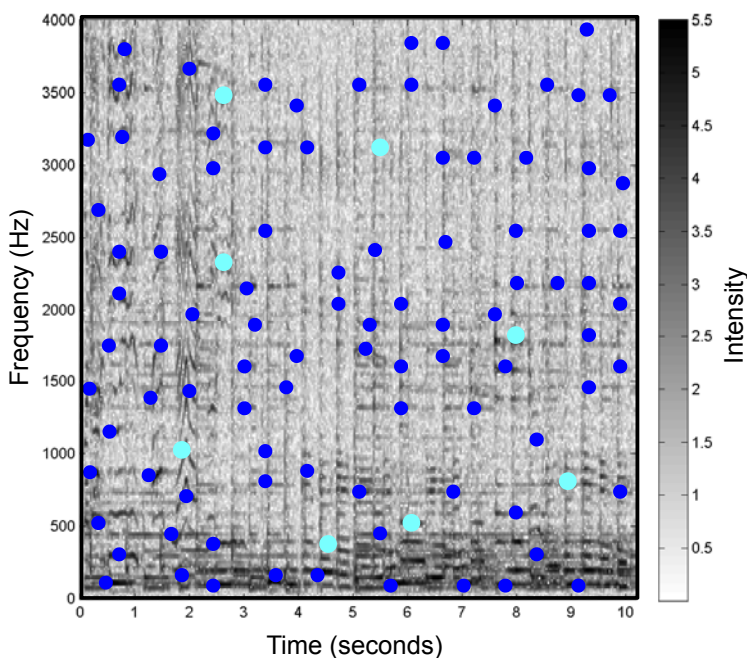
Fingerprints (Shazam)



Steps:

1. Spectrogram
2. Peaks

Fingerprints (Shazam)



Steps:

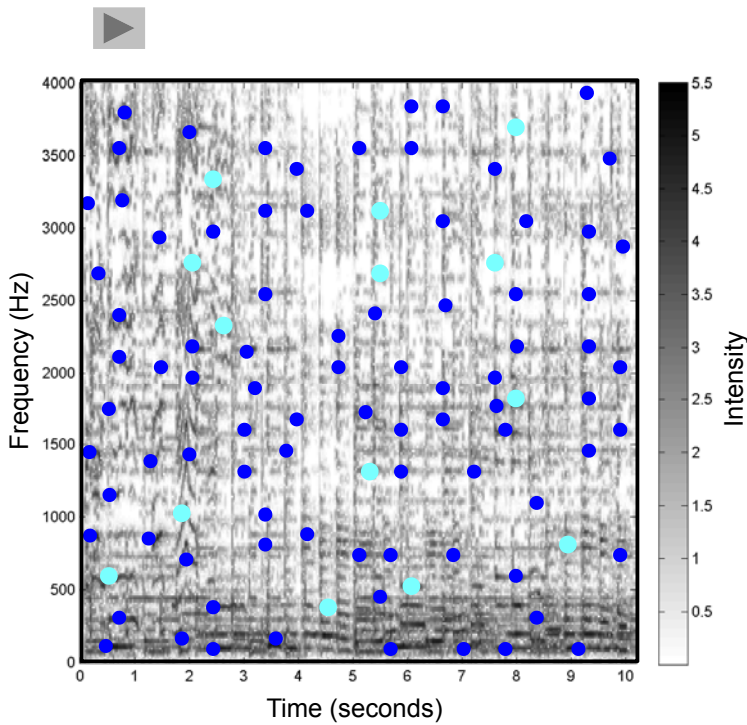
1. Spectrogram
2. Peaks / differing peaks

Robustness:

- Noise, reverb, room acoustics, equalization



Fingerprints (Shazam)



Steps:

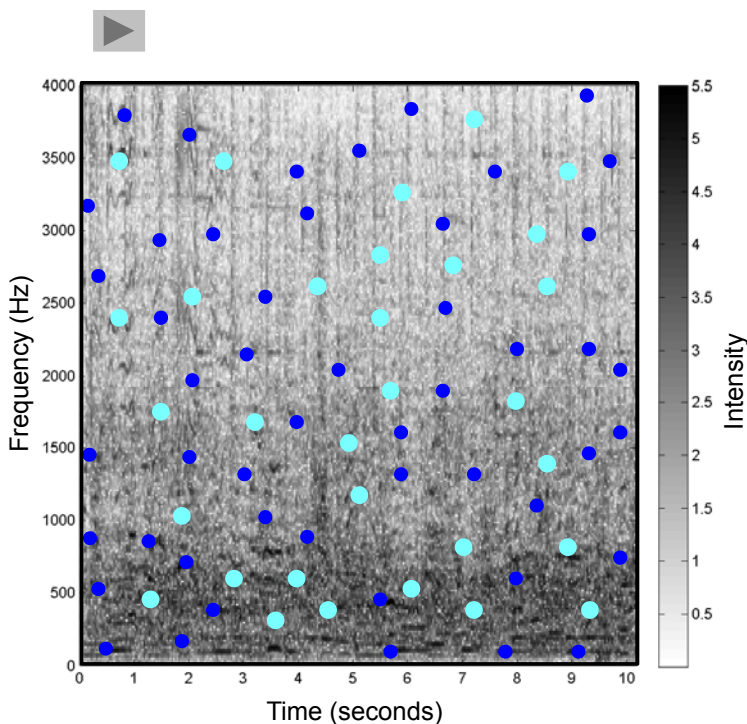
1. Spectrogram
2. Peaks / differing peaks

Robustness:

- Noise, reverb, room acoustics, equalization
- Audio codec



Fingerprints (Shazam)



Steps:

1. Spectrogram
2. Peaks / differing peaks

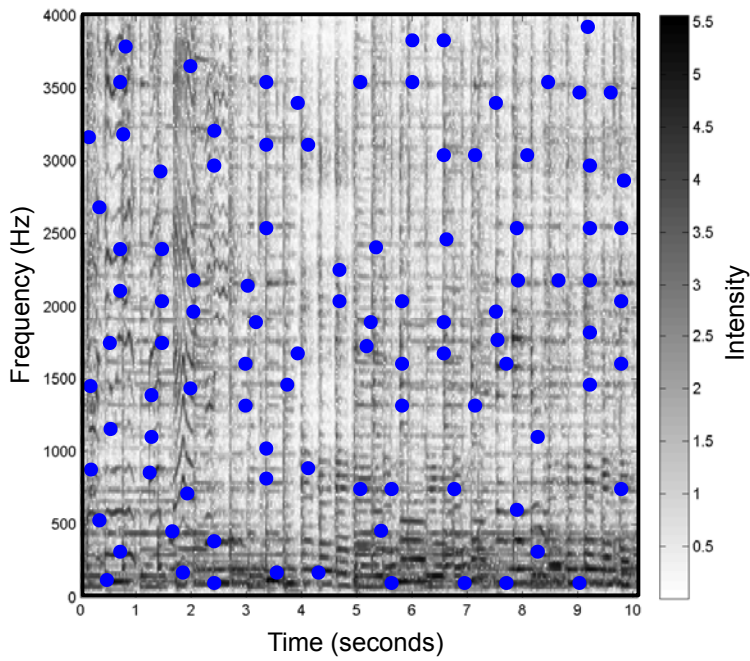
Robustness:

- Noise, reverb, room acoustics, equalization
- Audio codec
- Superposition of other audio sources



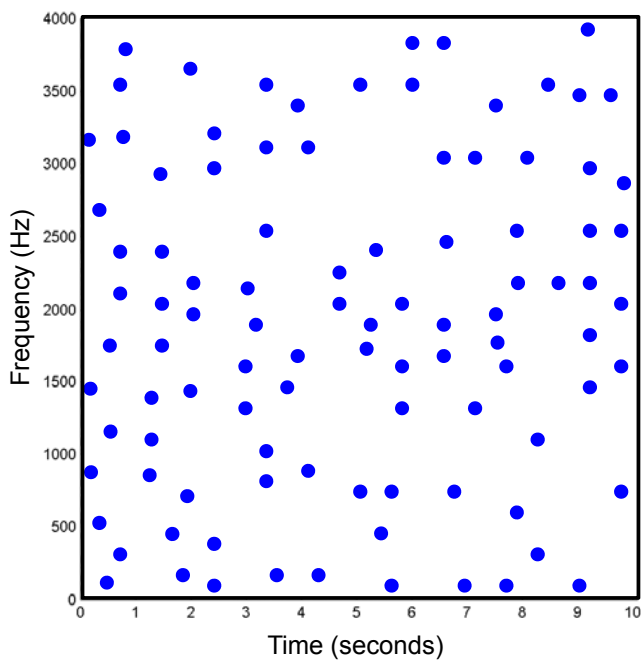
Matching Fingerprints (Shazam)

Database document



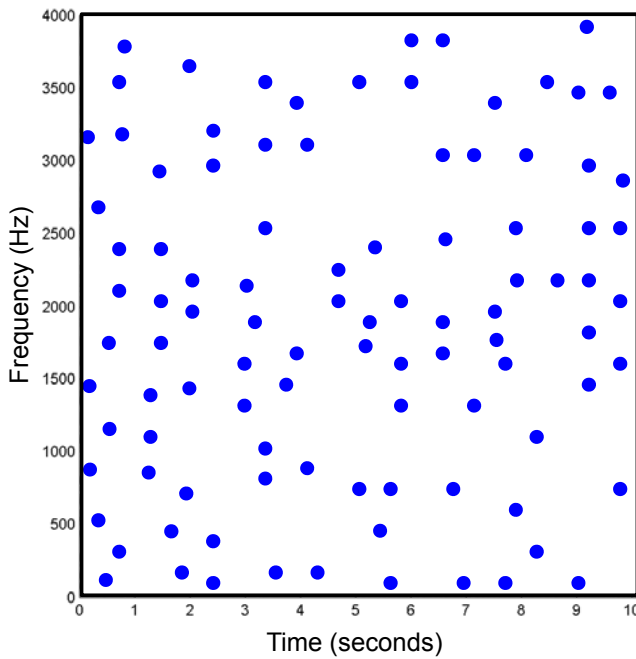
Matching Fingerprints (Shazam)

Database document
(constellation map)

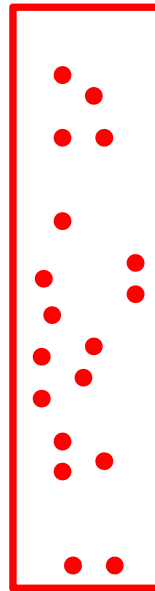


Matching Fingerprints (Shazam)

Database document
(constellation map)

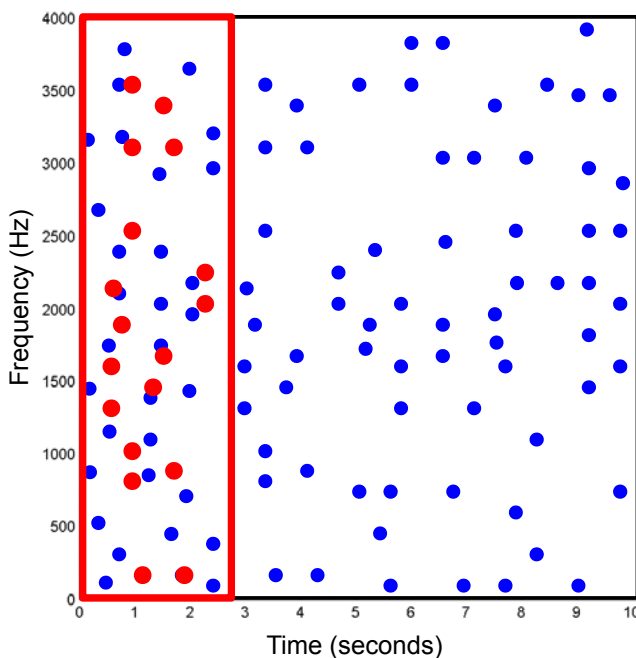


Query document
(constellation map)



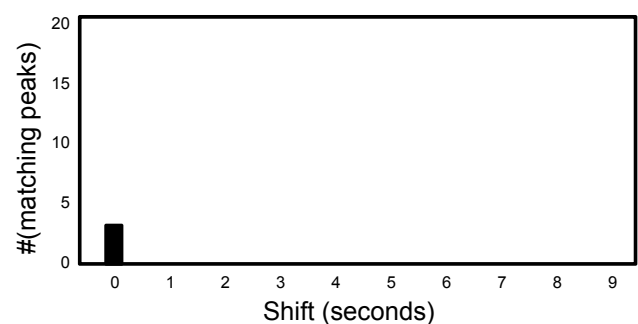
Matching Fingerprints (Shazam)

Database document
(constellation map)



Query document
(constellation map)

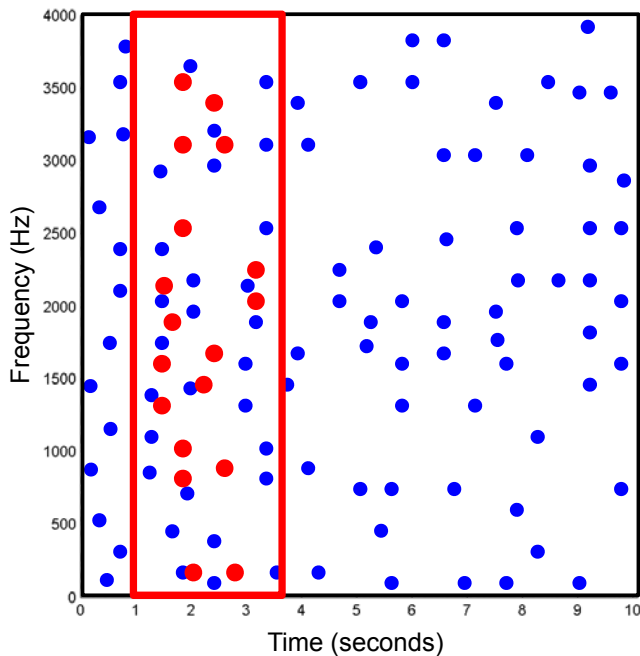
1. Shift query across database document
2. Count matching peaks



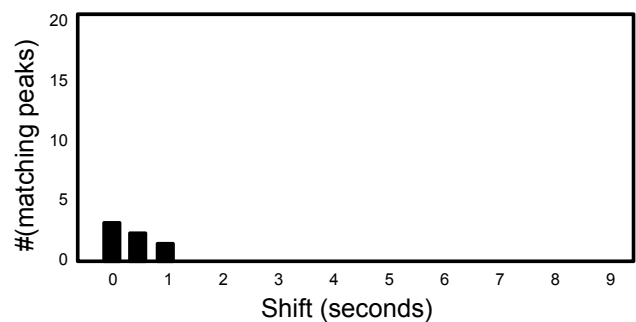
Matching Fingerprints (Shazam)

Database document
(constellation map)

Query document
(constellation map)



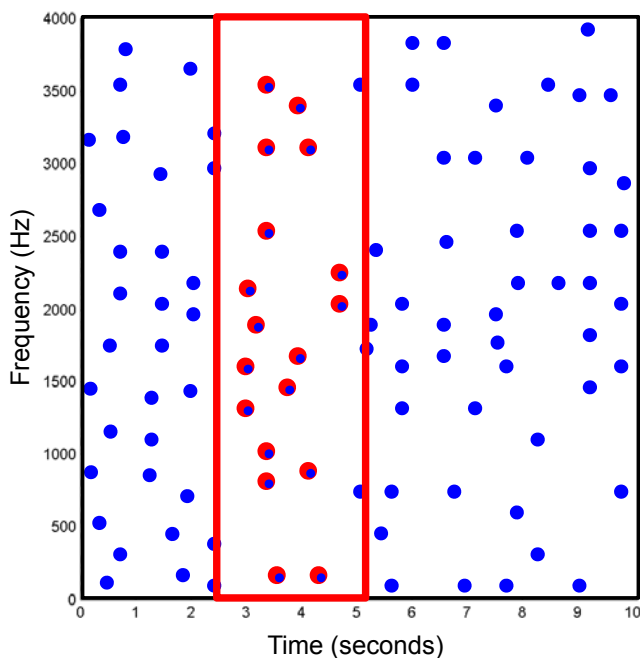
1. Shift query across database document
2. Count matching peaks



Matching Fingerprints (Shazam)

Database document
(constellation map)

Query document
(constellation map)



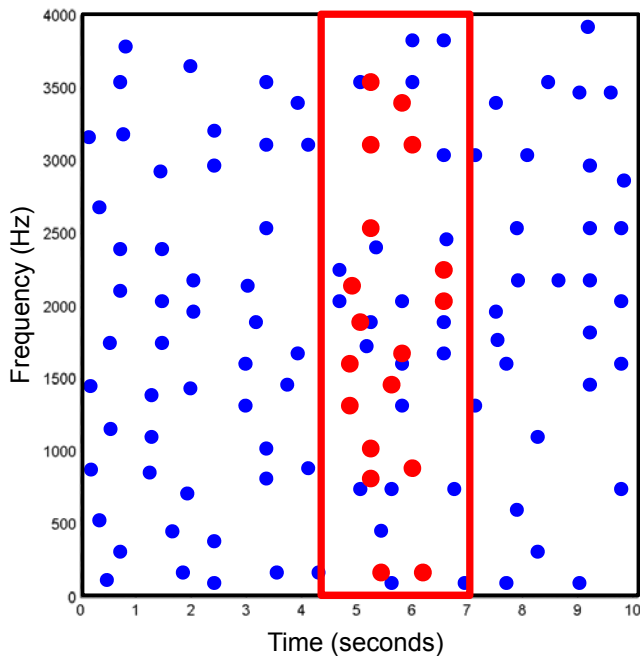
1. Shift query across database document
2. Count matching peaks



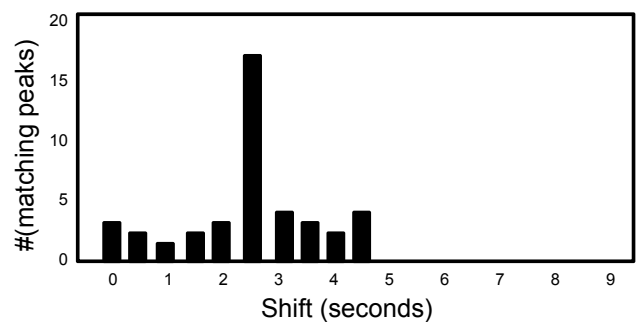
Matching Fingerprints (Shazam)

Database document
(constellation map)

Query document
(constellation map)



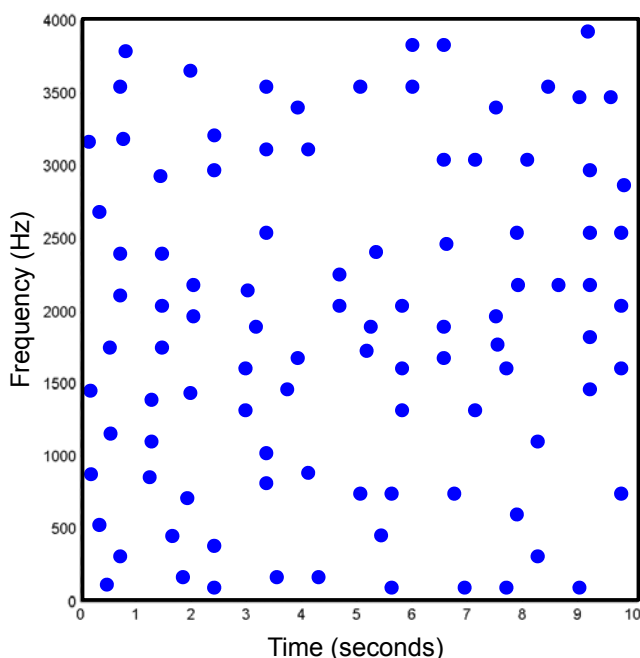
1. Shift query across database document
2. Count matching peaks



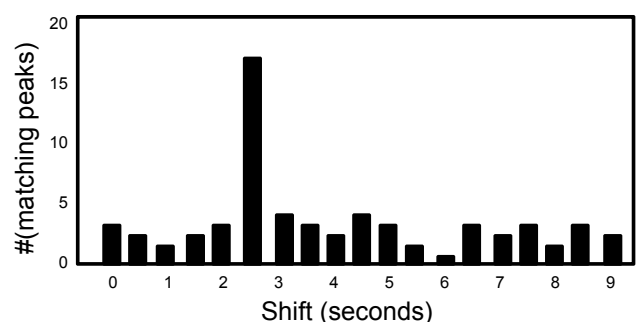
Matching Fingerprints (Shazam)

Database document
(constellation map)

Query document
(constellation map)

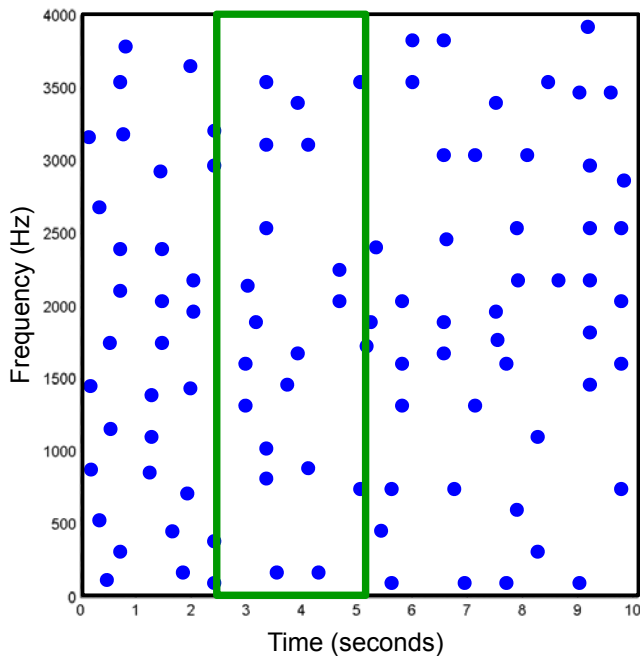


1. Shift query across database document
2. Count matching peaks



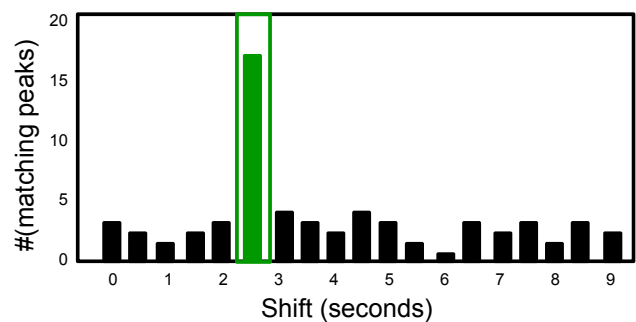
Matching Fingerprints (Shazam)

Database document
(constellation map)



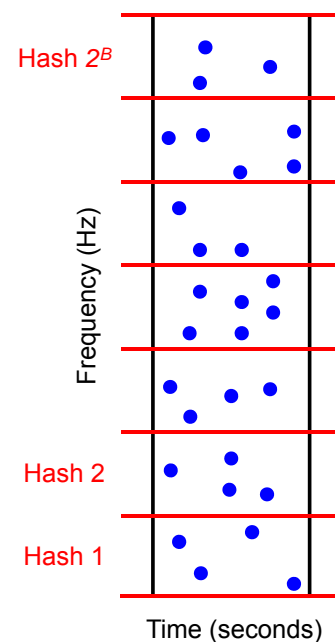
Query document
(constellation map)

1. Shift query across database document
2. Count matching peaks
3. High count indicates a hit (document ID & position)



Indexing (Shazam)

- Index the fingerprints using hash lists
- Hashes correspond to (quantized) frequencies

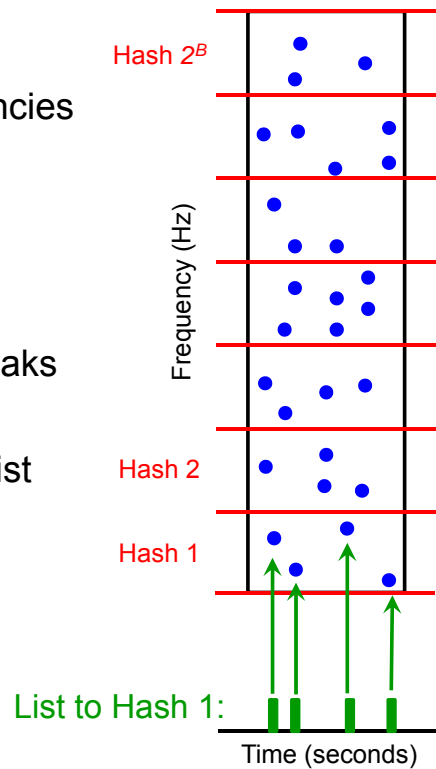


Indexing (Shazam)

- Index the fingerprints using hash lists
 - **Hashes** correspond to (quantized) frequencies
 - **Hash list** consists of time positions (and document IDs)
-
- N = number of spectral peaks
 - B = #(bits) used to encode spectral peaks
 - 2^B = number of hash lists
 - $N / 2^B$ = average number of elements per list

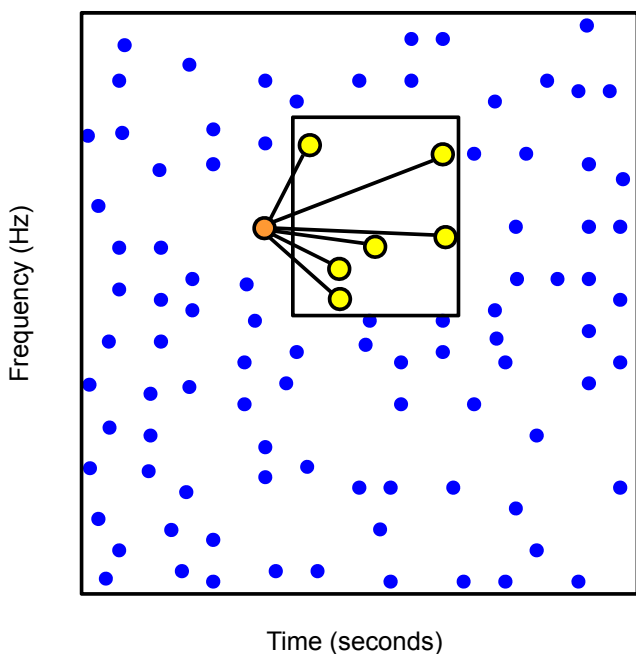
Problem:

- Individual peaks are not characteristic
- Hash lists may be very long
- Not suitable for indexing



Indexing (Shazam)

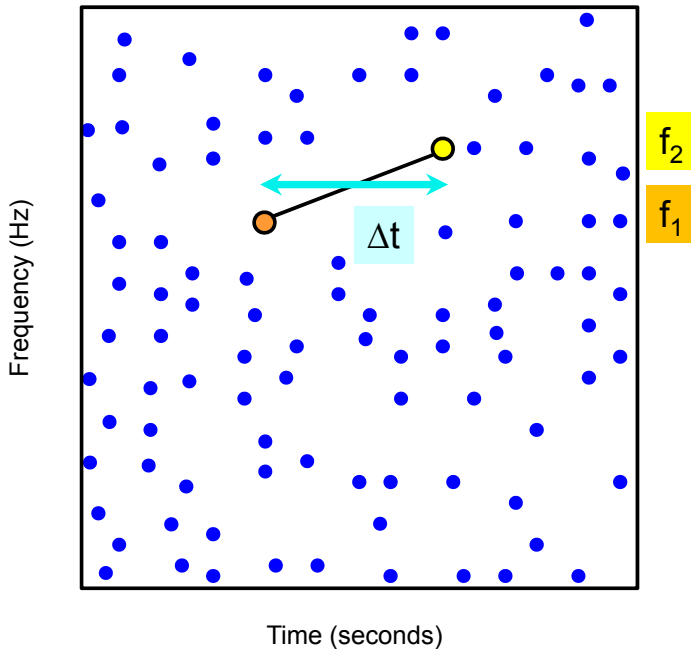
Idea: Use pairs of peaks to increase specificity of hashes



1. Peaks
2. Fix anchor point
3. Define target zone
4. Use pairs of points
5. Use every point as anchor point

Indexing (Shazam)

Idea: Use pairs of peaks to increase specificity of hashes



1. Peaks
2. Fix anchor point
3. Define target zone
4. Use pairs of points
5. Use every point as anchor point

New hash:

Consists of two frequency values and a time difference:

$$(f_1, f_2, \Delta t)$$

Indexing (Shazam)

- A hash is formed between an anchor point and each point in the target zone using two frequency values and a time difference.
- Fan-out (taking pairs of peaks) may cause a combinatorial explosion in the number of tokens. However, this can be controlled by the size of the target zone.
- Using more complex hashes increases specificity (leading to much smaller hash lists) and speed (making the retrieval much faster).

Indexing (Shazam)

Definitions:

- N = number of spectral peaks
- p = probability that a spectral peak can be found in (noisy and distorted) query
- F = fan-out of target zone, e. g. $F = 10$
- B = #(bits) used to encode spectral peaks and time difference

Consequences:

- $F \cdot N$ = #(tokens) to be indexed
- 2^{B+B} = increase of specificity (2^{B+B+B} instead of 2^B)
- p^2 = probability of a hash to survive
- $p \cdot (1 - (1-p)^F)$ = probability that, at least, one hash survives per anchor point

Example: $F = 10$ and $B = 10$

- Memory requirements: $F \cdot N = 10 \cdot N$
- Speedup factor: $2^{B+B} / F^2 \sim 10^6 / 10^2 = 10000$
(F times as many tokens in query and database, respectively)

Conclusions (Shazam)

Many parameters to choose:

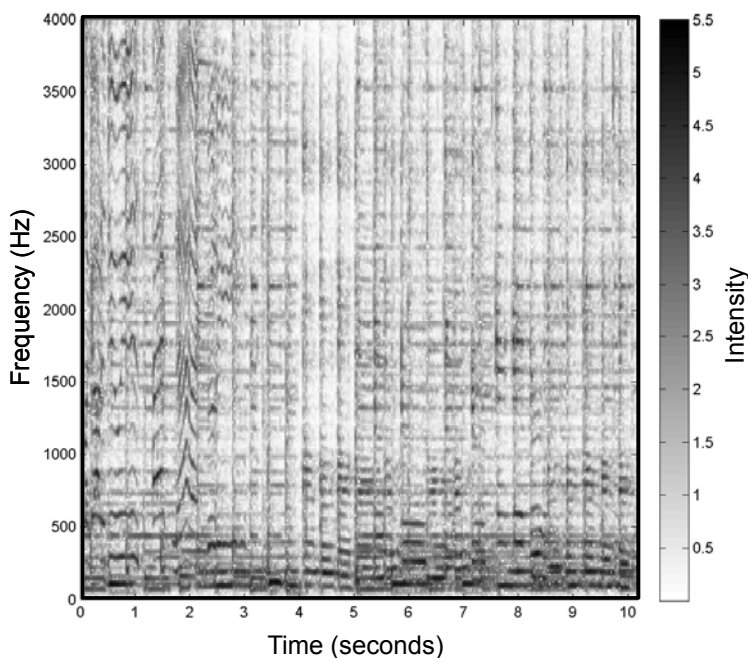
- Temporal and spectral resolution in spectrogram
- Peak picking strategy
- Target zone and fan-out parameter
- Hash function
- ...

Literature (Audio Identification)

- Allamanche et al. (AES 2001)
- Cano et al. (AES 2002)
- **Haitsma/Kalker (ISMIR 2002)**
- Kurth/Clausen/Ribbrock (AES 2002)
- Wang (ISMIR 2003)
- ⋮
- Dupraz/Richard (ICASSP 2010)
- Ramona/Peeters (ICASSP 2011)

PHILIPS

Fingerprints (Philips)

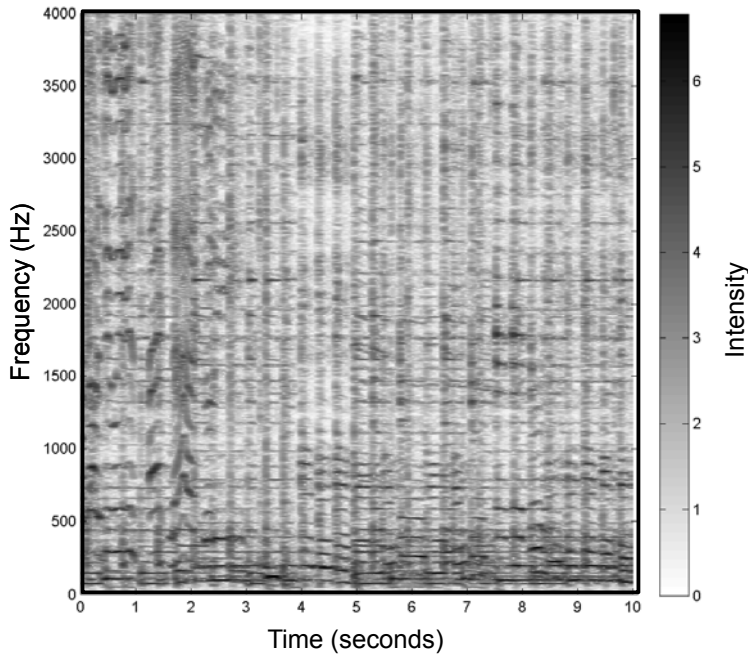


Steps:

1. Spectrogram

- *Efficiently computable*
- *Standard transform*
- *Robust*

Fingerprints (Philips)

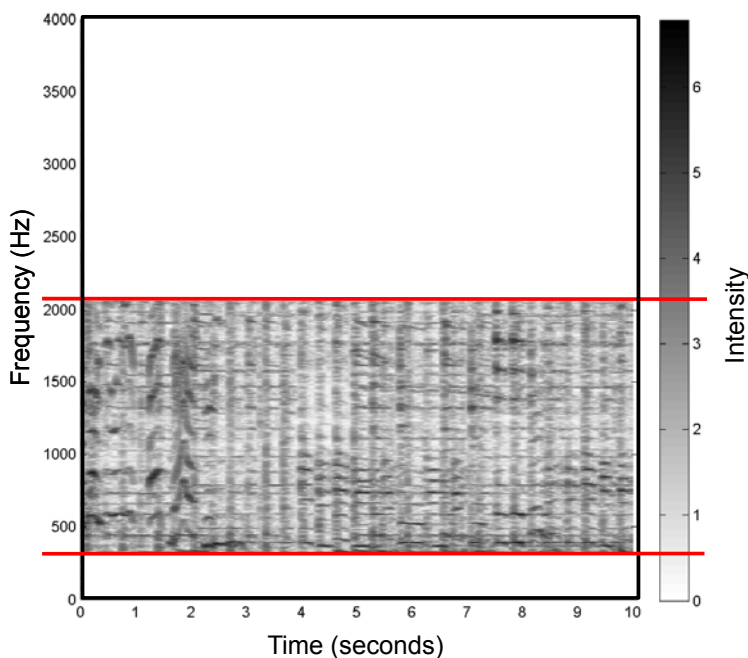


Steps:

1. Spectrogram (long window)

- *Coarse temporal resolution*
- *Large overlap of windows*
- *Robust to temporal distortion*

Fingerprints (Philips)

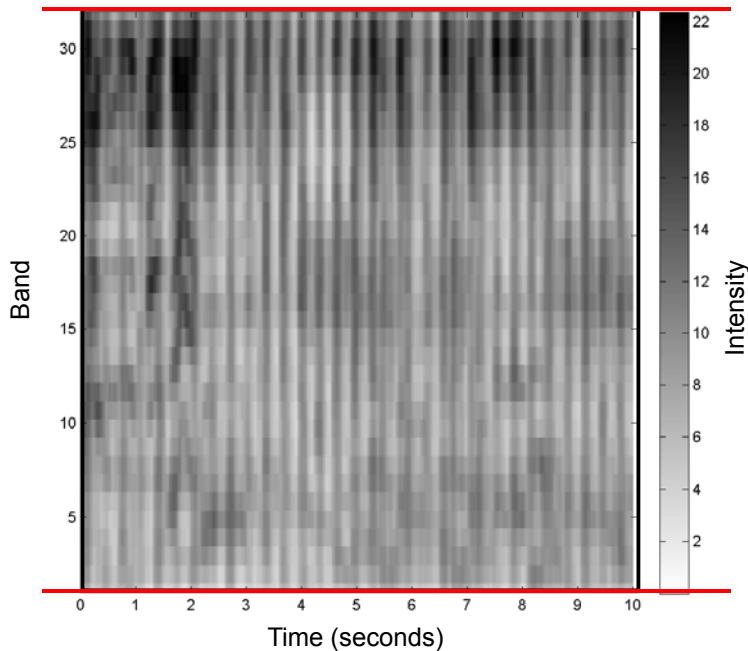


Steps:

1. Spectrogram (long window)
2. Consider limited frequency range

- *300 – 2000 Hz*
- *Most relevant spectral range (perceptually)*

Fingerprints (Philips)

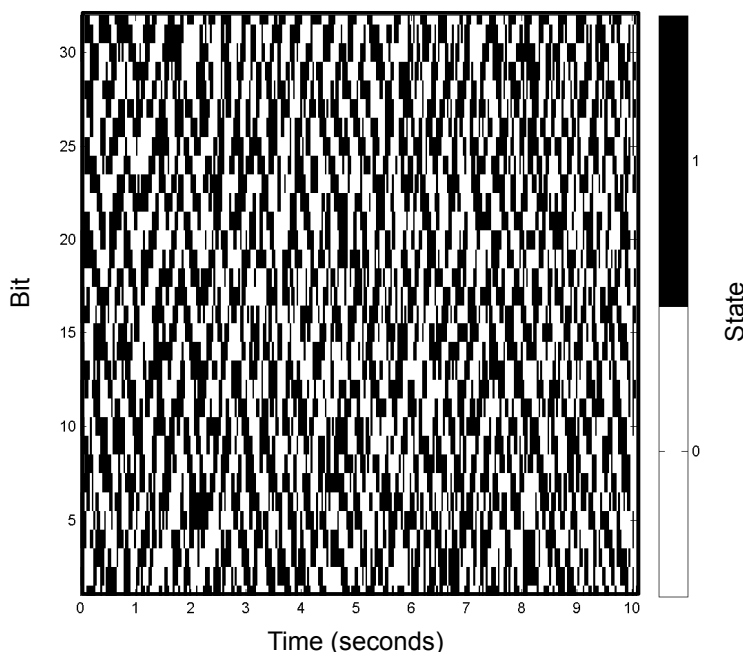


Steps:

1. Spectrogram (long window)
2. Consider limited frequency range
3. Log-frequency (Bark scale)

- 300 – 2000 Hz
- Most relevant spectral range (perceptually)
- 33 bands (roughly bark scale)
- Coarse frequency resolution
- Robust to spectral distortions

Fingerprints (Philips)

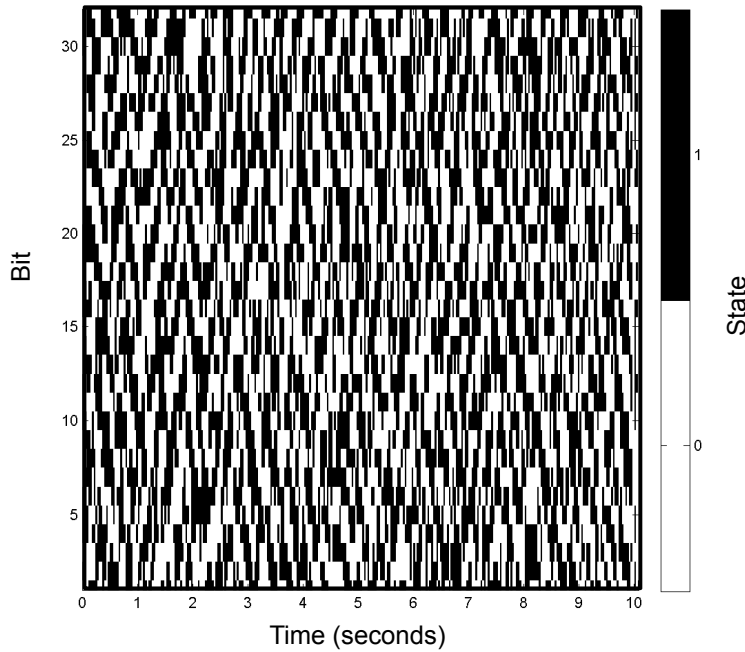


Steps:

1. Spectrogram (long window)
2. Consider limited frequency range
3. Log-frequency (Bark scale)
4. Binarization

- Local thresholding
- Sign of energy difference (simultaneously along time and frequency axes)
- Sequence of 32-bit vectors

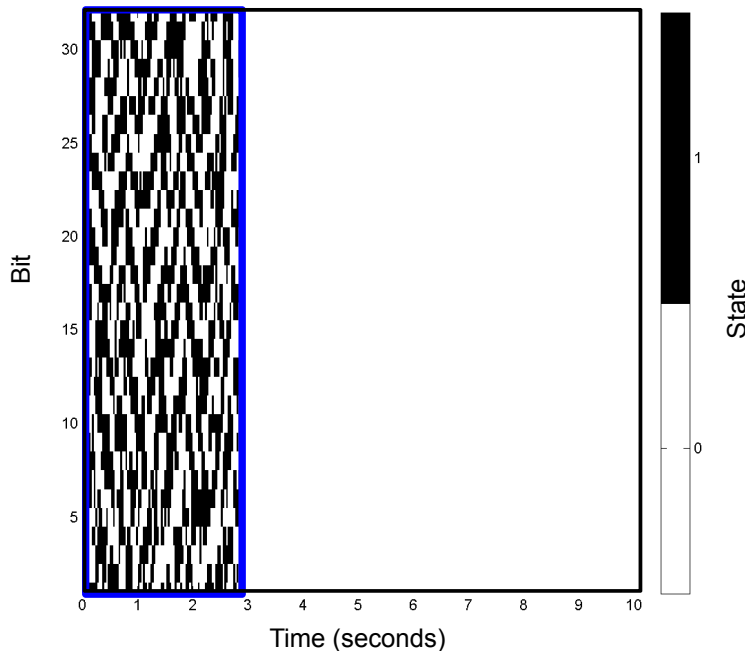
Fingerprints (Philips)



Sub-fingerprint:

- 32-bit vector
- Not characteristic enough

Fingerprints (Philips)



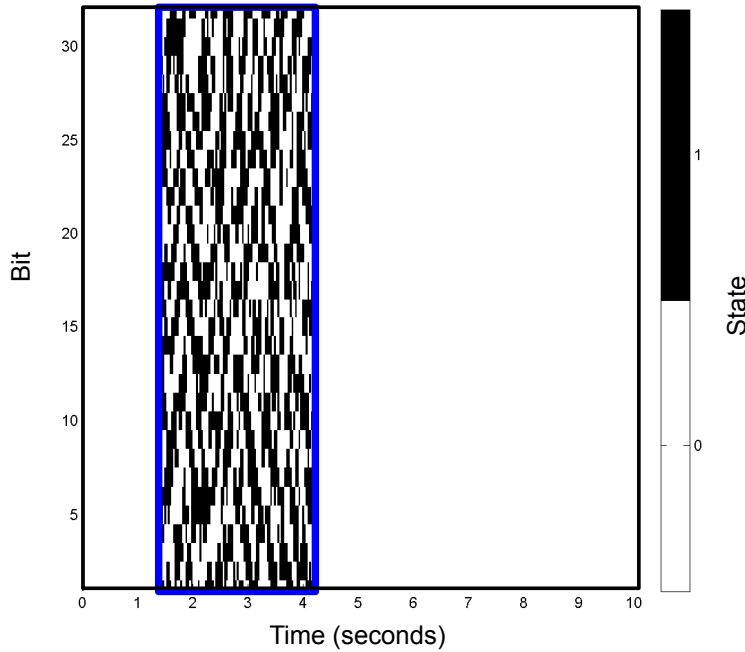
Sub-fingerprint:

- 32-bit vector
- Not characteristic enough

Fingerprint-block:

- 256 consecutive sub-fingerprints
- Covers roughly 3 seconds
- Overlapping

Fingerprints (Philips)



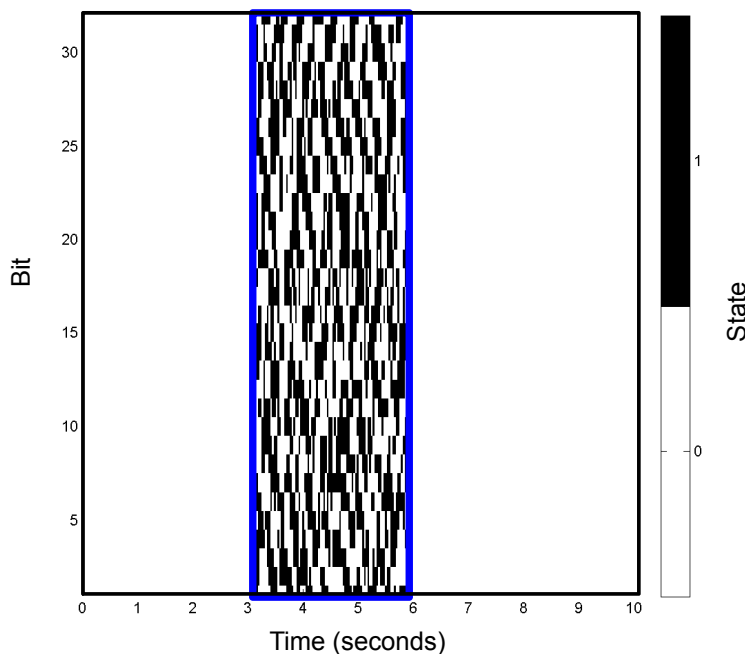
Sub-fingerprint:

- 32-bit vector
- Not characteristic enough

Fingerprint-block:

- 256 consecutive sub-fingerprints
- Covers roughly 3 seconds
- Overlapping

Fingerprints (Philips)



Sub-fingerprint:

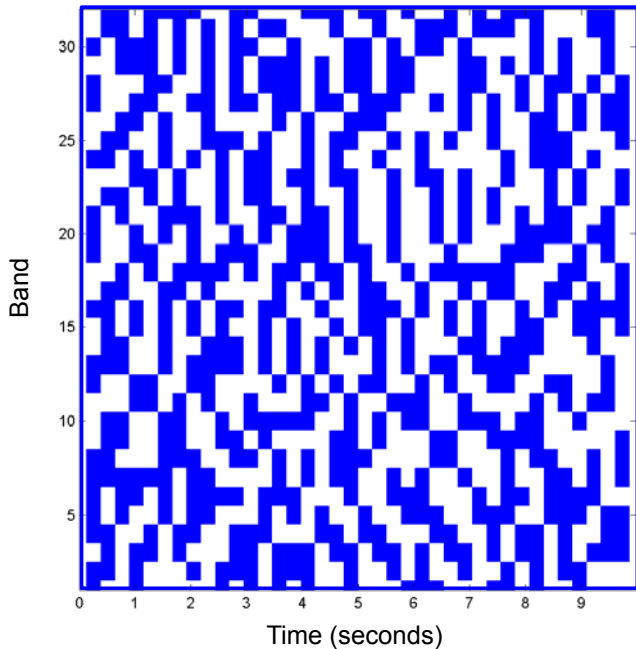
- 32-bit vector
- Not characteristic enough

Fingerprint-block:

- 256 consecutive sub-fingerprints
- Covers roughly 3 seconds
- Overlapping

Matching Fingerprints (Philips)

Database document
(fingerprint-blocks)

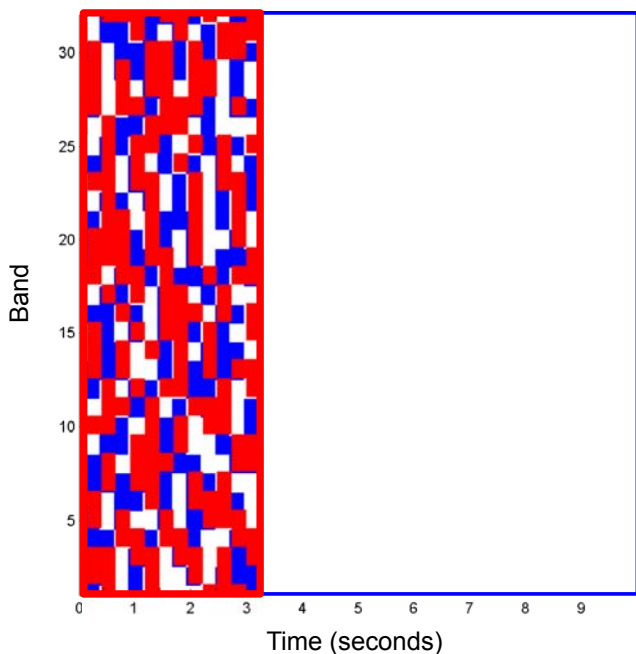


Query document
(fingerprint-block)



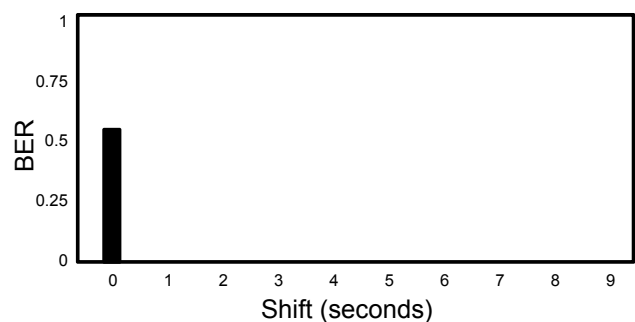
Matching Fingerprints (Philips)

Database document
(fingerprint-blocks)



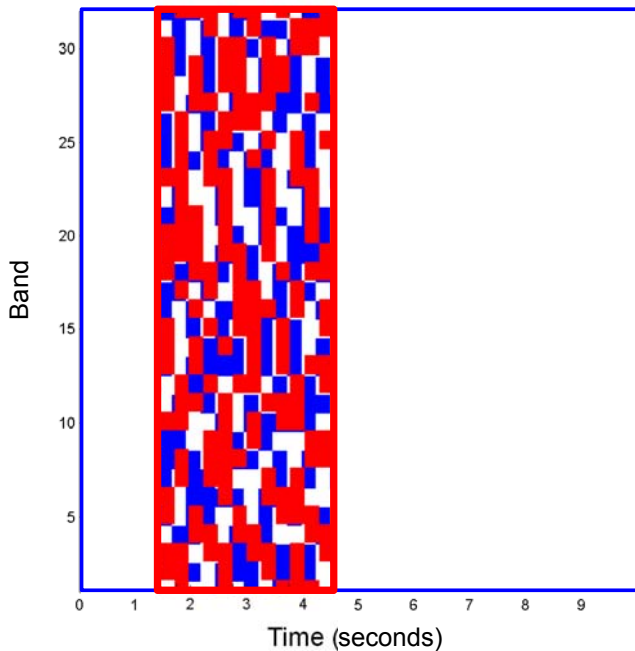
Query document
(fingerprint-block)

1. Shift query across database document
2. Calculate a block-wise bit-error-rate (BER)



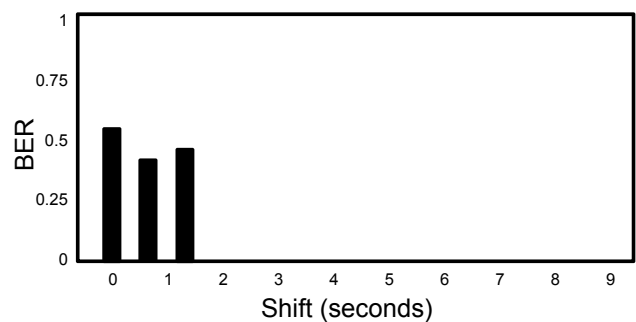
Matching Fingerprints (Philips)

Database document
(fingerprint-blocks)



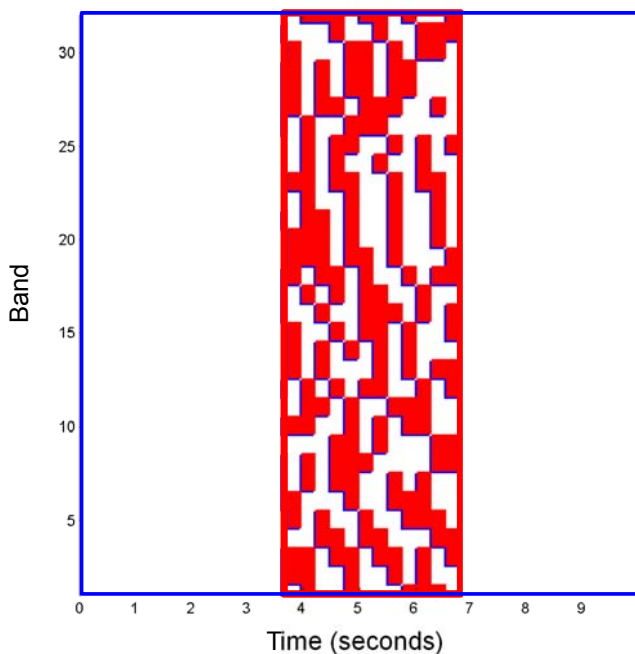
Query document
(fingerprint-block)

1. Shift query across database document
2. Calculate a block-wise bit-error-rate (BER)



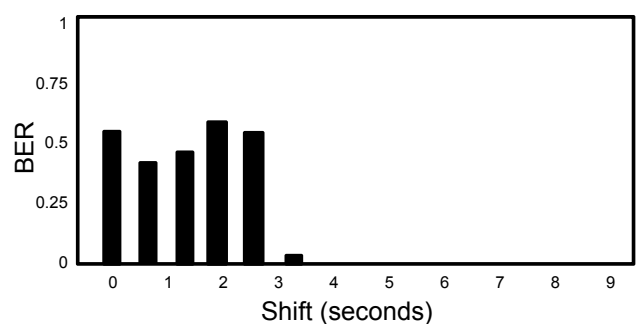
Matching Fingerprints (Philips)

Database document
(fingerprint-blocks)



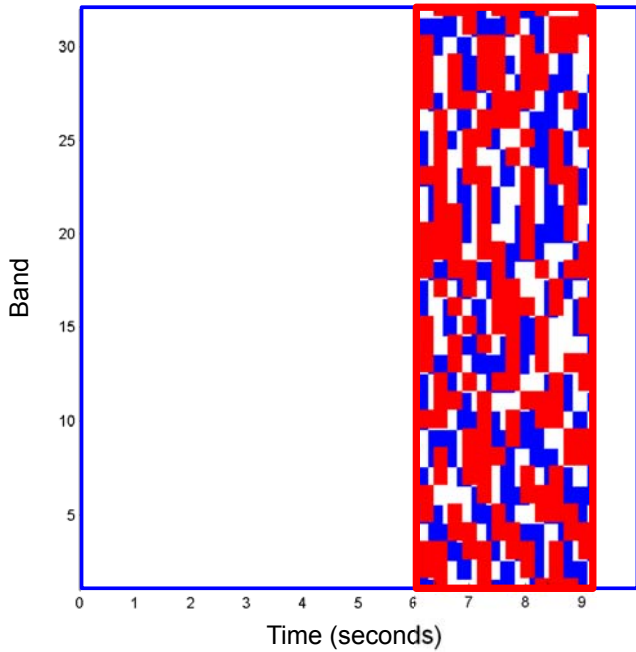
Query document
(fingerprint-block)

1. Shift query across database document
2. Calculate a block-wise bit-error-rate (BER)



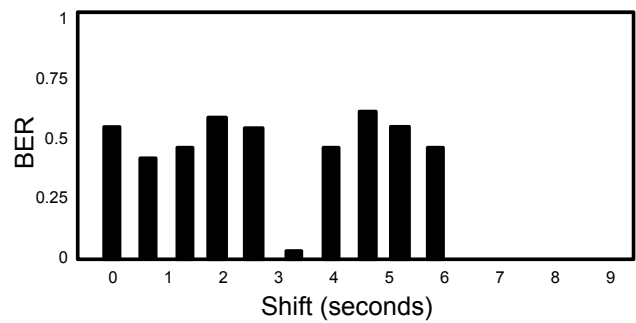
Matching Fingerprints (Philips)

Database document
(fingerprint-blocks)



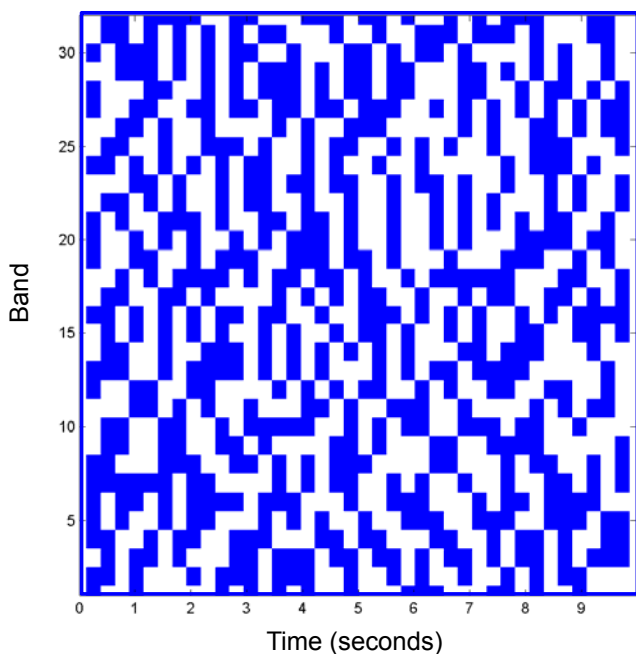
Query document
(fingerprint-block)

1. Shift query across database document
2. Calculate a block-wise bit-error-rate (BER)



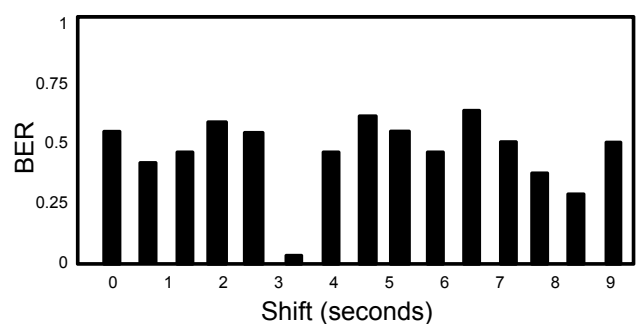
Matching Fingerprints (Philips)

Database document
(fingerprint-blocks)



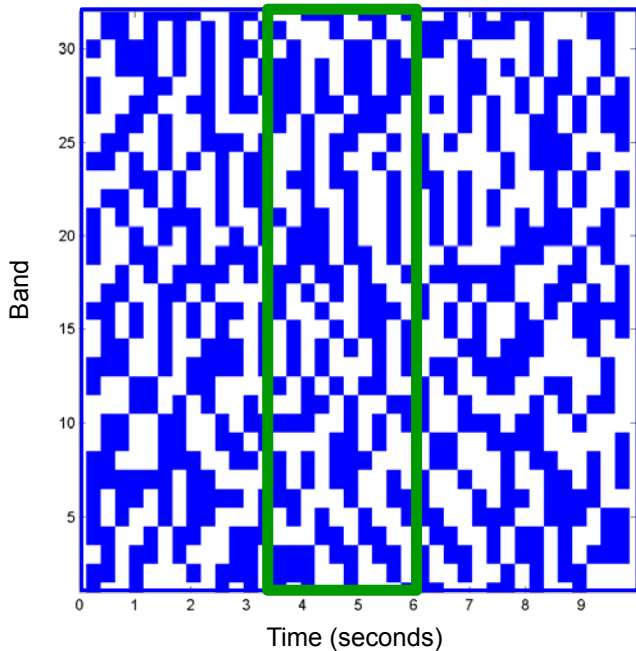
Query document
(fingerprint-block)

1. Shift query across database document
2. Calculate a block-wise bit-error-rate (BER)



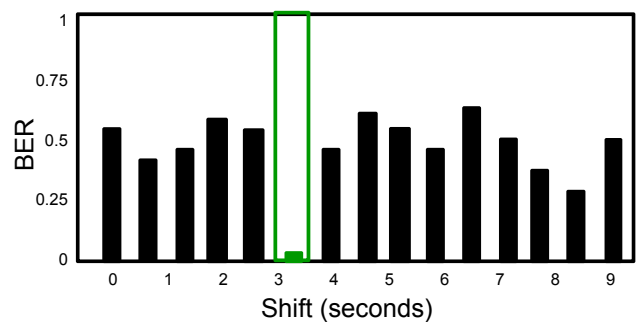
Matching Fingerprints (Philips)

Database document
(fingerprint-blocks)



Query document
(fingerprint-block)

1. Shift query across database document
2. Calculate a block-wise bit-error-rate (BER)
3. Low BER indicates hit



Indexing (Philips)

Note:

- Individual sub-fingerprints (32 bit) are not characteristic
- Fingerprint blocks (256 sub-fingerprints, 8 kbit) are used

Problem:

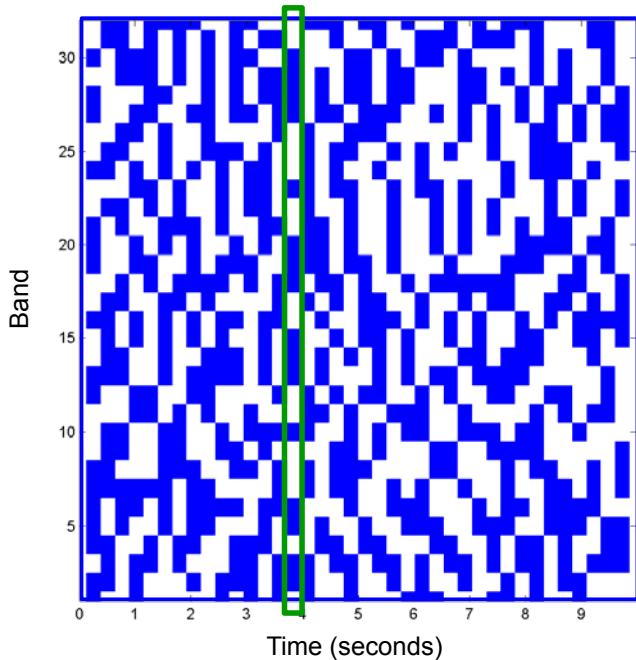
- Computation of BER between **query fingerprint-block** and every **database fingerprint-block** is expensive
- Chance that a complete fingerprint-block survives is low
- Exact hashing problematic

Strategy:

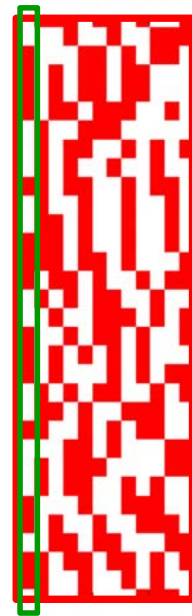
- Only sub-fingerprints are indexed using hashing
- Exact sub-fingerprint matches are used to identify **candidate fingerprint-blocks** in database.
- BER is only computed between **query fingerprint-block** and **candidate fingerprint-blocks**
- Procedure is terminated when **database fingerprint-block** is found, where BER falls below a certain threshold

Indexing (Philips)

Database document
(fingerprint-blocks)



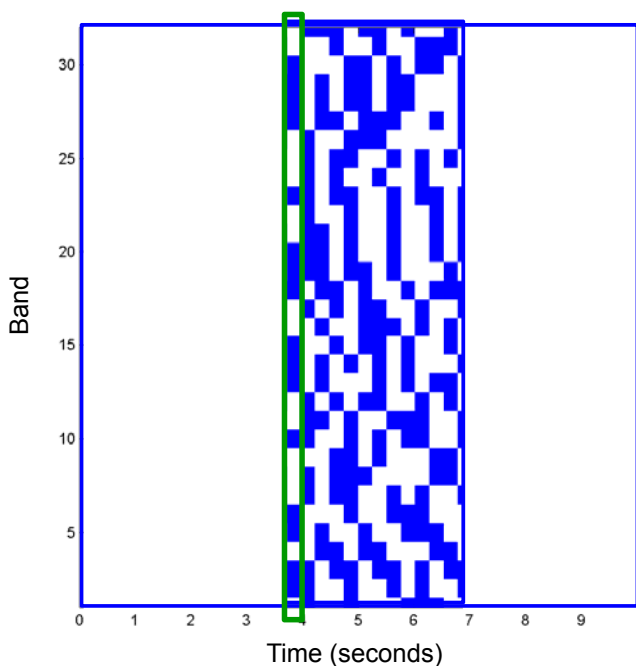
Query document
(fingerprint-block)



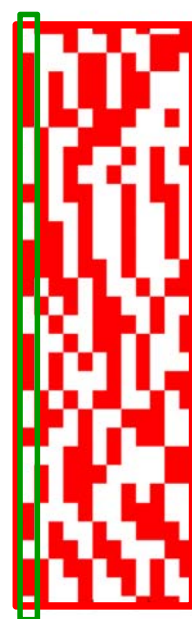
1. Efficient search for **exact matches** of sub-fingerprints (anchor points)

Indexing (Philips)

Database document
(fingerprint-blocks)



Query document
(fingerprint-block)



1. Efficient search for **exact matches** of sub-fingerprints (anchor points)
2. Calculate BER only for blocks containing anchor points

Conclusions (Philips)

- Comparing binary fingerprint-blocks expressing tempo-spectral changes
- Usage of some sort of shingling technique
 - see [Casey et al. 2008, IEEE-TASLP] for a similar approach applied to a more general retrieval task
- Acceleration using hash-based search for anchor-points (sub-fingerprints)
- Concepts of fault tolerance are required to increase robustness
- Susceptible to distortions in specific frequency bands (e. g. equalization) or to superpositions with other sources

Conclusions (Audio Identification)

- Basic techniques used in Shazam and Philip systems
- Many more ways to define robust audio fingerprints
- Delicate trade-off between specificity, robustness, and efficiency
- Audio recording is identified (**not** a piece of music)
- Does not allow for identifying studio recording using a query taken from live recordings
- Does not generalize to identify different interpretations or versions of the same piece of music

Overview (Audio Retrieval)

- Audio identification (audio fingerprinting)
- **Audio matching**
- Cover song identification



Audio Matching

- Database:** Audio collection containing:
- Several recordings of the same piece of music
 - Different interpretations by various musicians
 - Arrangements in different instrumentations
- Goal:** Given a short **query audio fragment**, find all corresponding audio fragments of similar musical content.
- Notes:**
- Instance of fragment-based retrieval
 - Medium specificity
 - A single document may contain several hits
 - Cross-modal retrieval also feasible

Audio Matching

Beethoven's Fifth



Various interpretations

Bernstein



Karajan



Scherbakov (piano)



MIDI (piano)



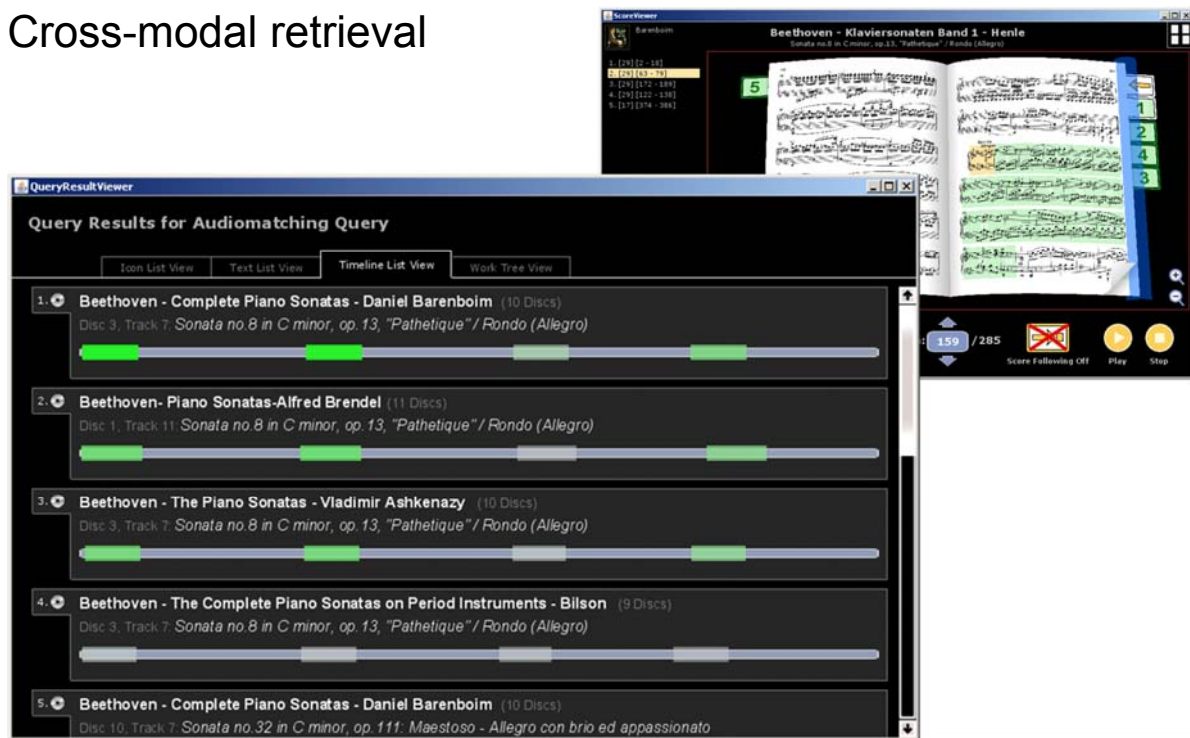
Application Scenario

Content-based retrieval

A screenshot of a software interface for audio matching. The left window, titled 'Plugin: Waveform Plotter/AudioMatching (Version 0.1, Build: Wed Jun 13 ...)', shows a waveform plot with a red section on the left and a green section on the right. Below the plot is a search results tree listing various audio files, including 'Beethoven_op067_1_symphony_5_bernstein_22050_mono.wav' and 'Beethoven_op067_1_symphony_5_karajan_22050_mono.wav'. The right window, titled 'ResultView: Switcher (Version 0.14)', displays a list of these files with their durations and playback controls. The first file, 'Beethoven_op067_1_symphony_5_bernstein_22050_mono.wav', is selected and has a duration of 08:38.00. The interface also shows a 'wav Directory' field and a 'Track ->17 Current Track Length: 08:38.00' indicator at the bottom.

Application Scenario

Cross-modal retrieval



Literature (Audio Matching)

- Pickens et al. (ISMIR 2002)
- Müller/Kurth/Clausen (ISMIR 2005)
- Suyoto et al. (IEEE TASLP 2008)
- Casey et al. (IEEE TASLP 2008)
- Kurth/Müller (IEEE TASLP 2008)
- Yu et al. (ACM MM 2010)
- ⋮

Audio Matching

Two main ingredients:

1.) Audio features

- Robust but discriminating
- Chroma-based features
- Correlate to harmonic progression
- Robust to variations in dynamics, timbre, articulation, local tempo

2.) Matching procedure

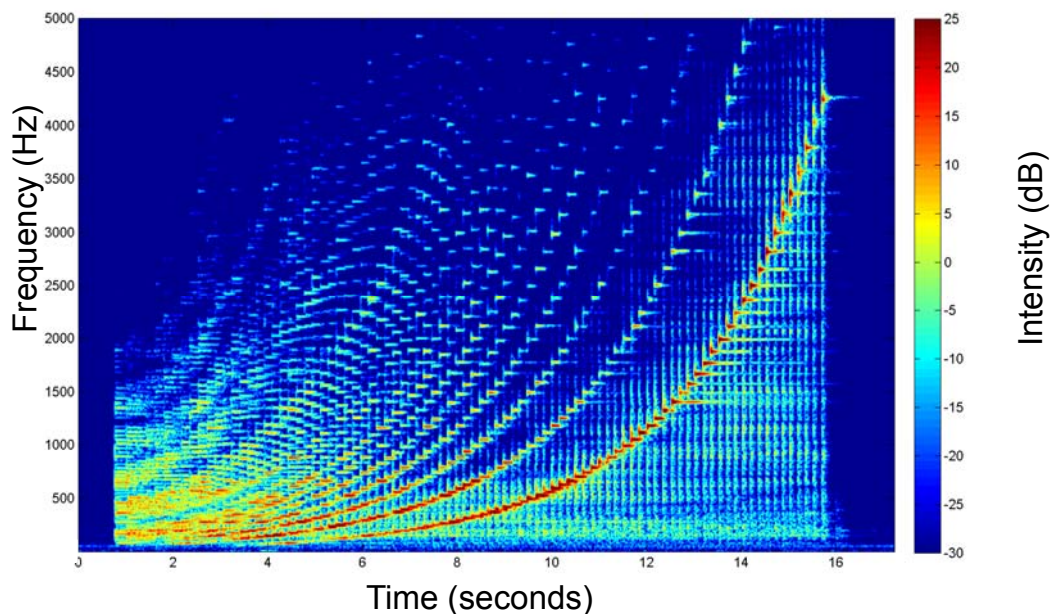
- Efficient
- Robust to local and global tempo variations
- Scalable using index structure

Audio Features

Example: Chromatic scale



Spectrogram

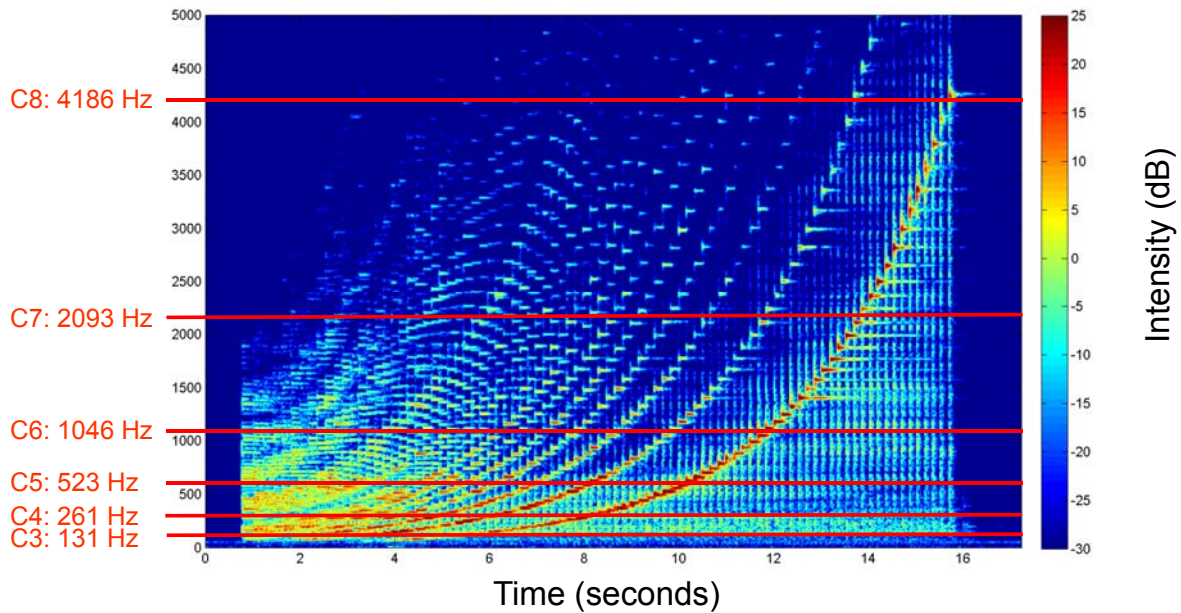


Audio Features

Example: Chromatic scale



Spectrogram

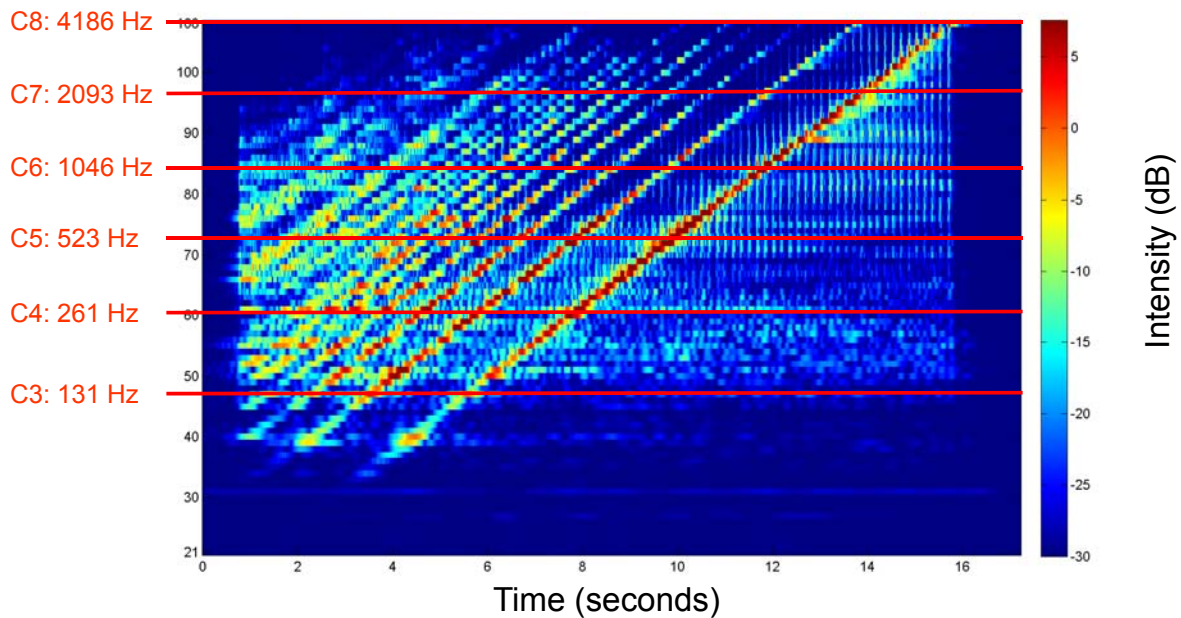


Audio Features

Example: Chromatic scale



Log-frequency spectrogram

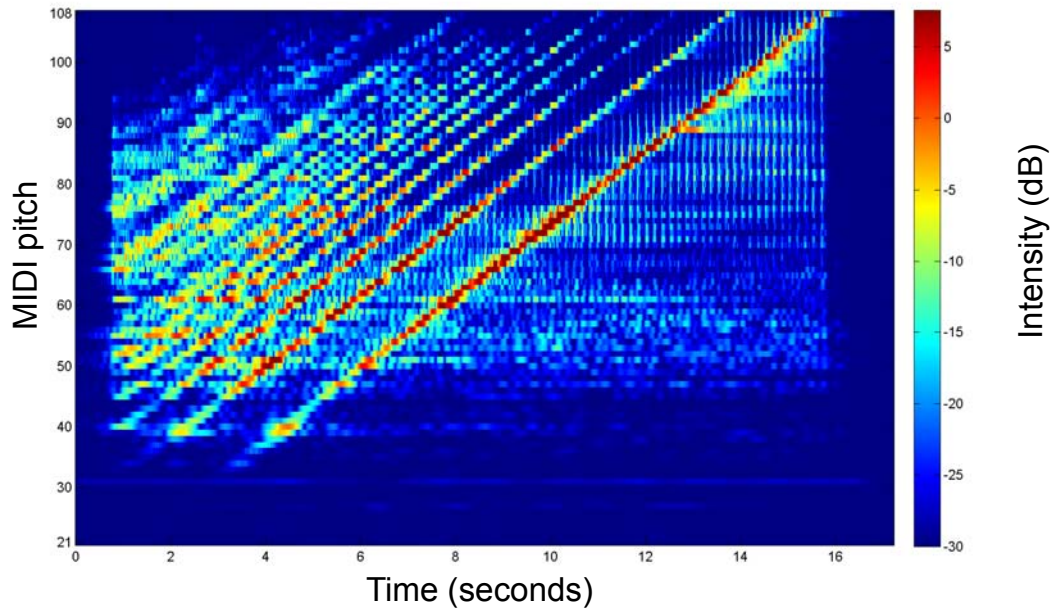


Audio Features

Example: Chromatic scale



Log-frequency spectrogram

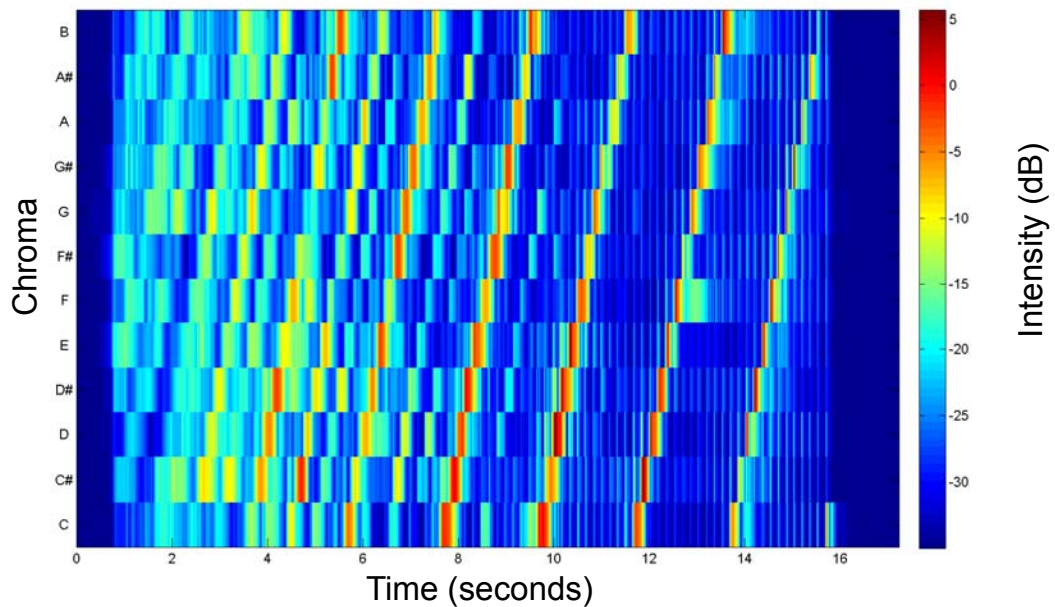


Audio Features

Example: Chromatic scale



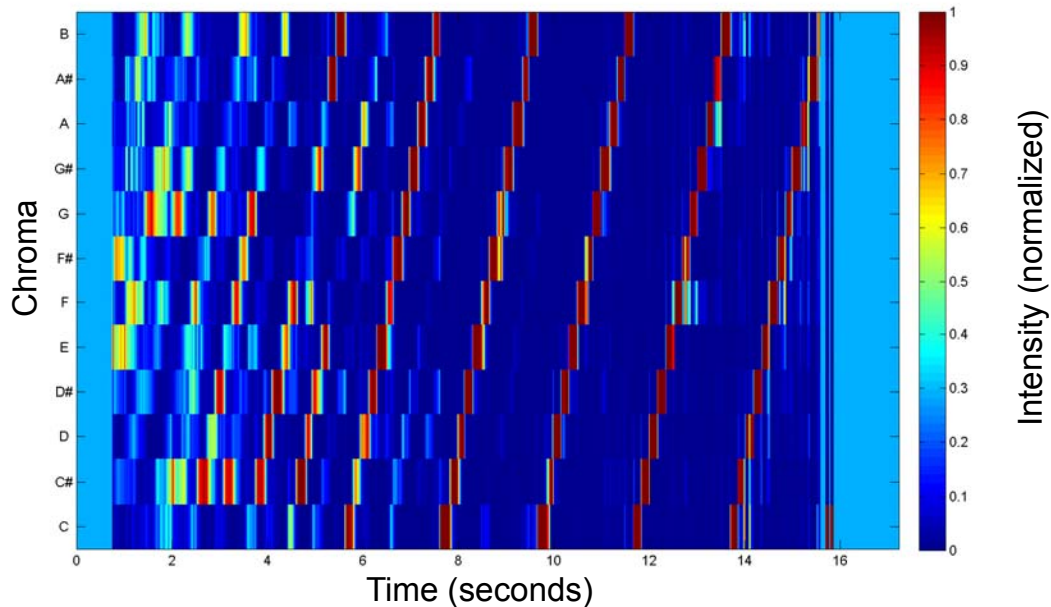
Chroma representation



Audio Features

Example: Chromatic scale

Chroma representation (normalized, Euclidean)



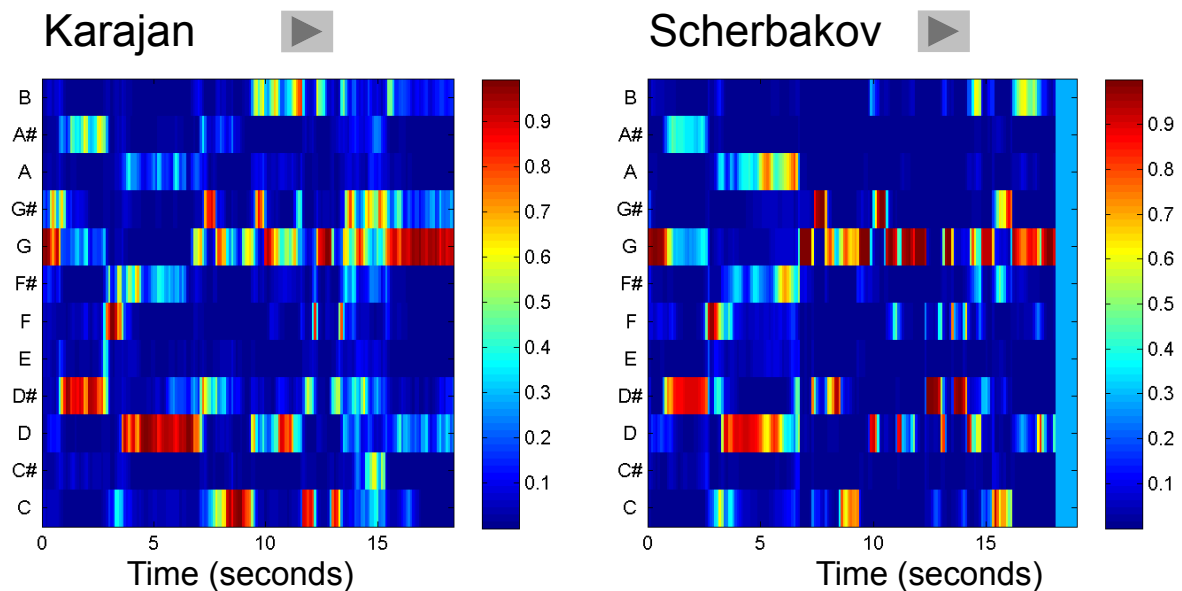
Audio Features

- Pitches are perceived as related (harmonically similar) if they differ by an octave
- Idea: through away information which is difficult to estimate and not so important for harmonic analysis
- Separation of pitch into two components:
tone height (octave number) and **chroma**
- Chroma: 12 traditional pitch classes of the equal-tempered scale. For example:
 $\text{Chroma } C \cong \{ \dots, C_0, C_1, C_2, C_3, \dots \}$
- Computation: pitch features \rightarrow chroma features
Add up all pitches belonging to the same class
- Result: 12-dimensional chroma vector

Audio Features

Example: Beethoven's Fifth

Chroma representation (normalized, 10 Hz)

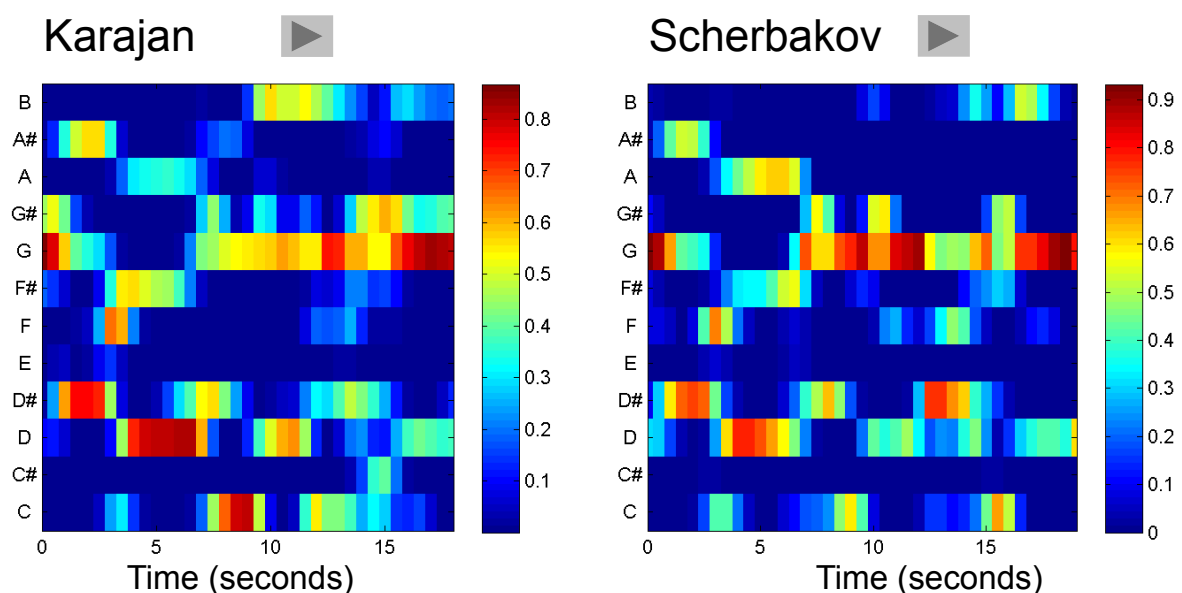


Audio Features

Example: Beethoven's Fifth

Chroma representation (normalized, 2 Hz)

Smoothing (2 seconds) + downsampling (factor 5)

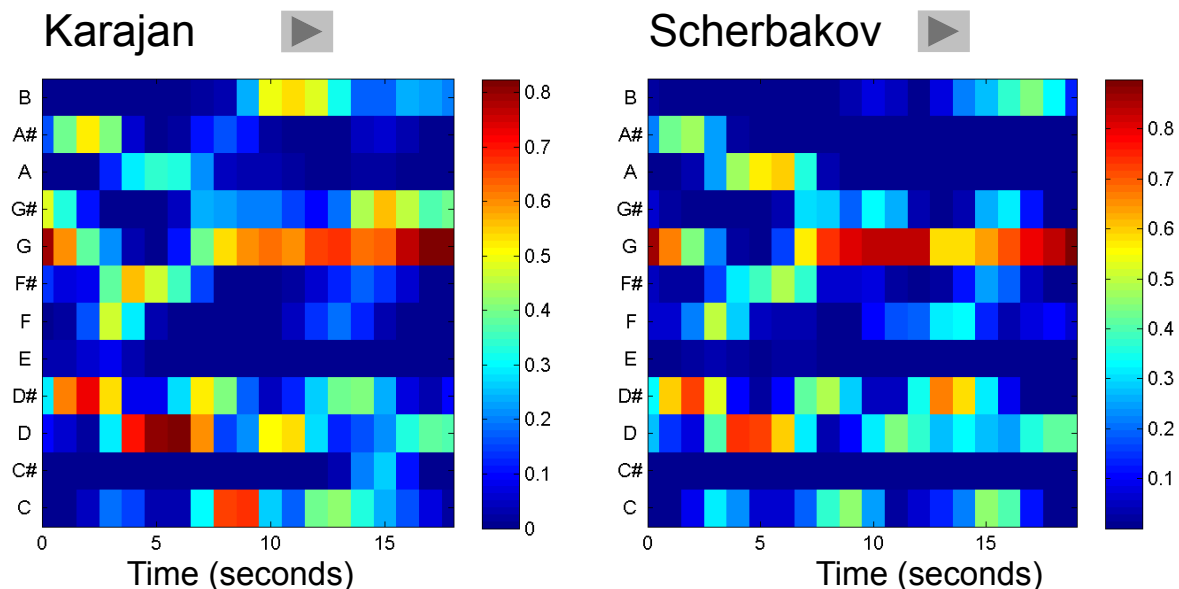


Audio Features

Example: Beethoven's Fifth

Chroma representation (normalized, 1 Hz)

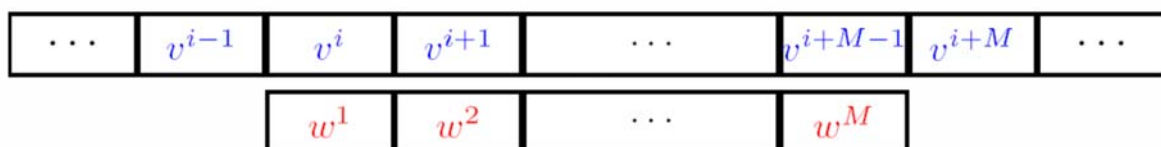
Smoothing (4 seconds) + downsampling (factor 10)



Matching Procedure

Compute chroma feature sequences

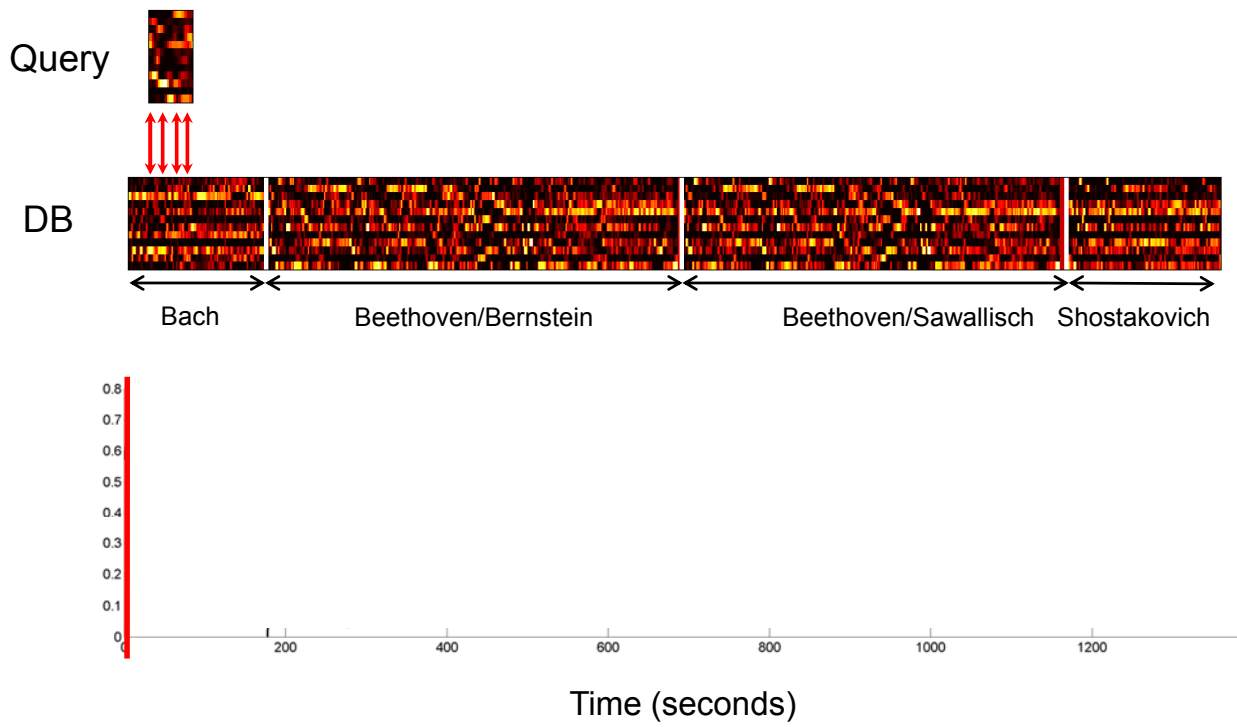
- Database $D \rightsquigarrow F[D] = (v^1, v^2, \dots, v^N)$
- Query $Q \rightsquigarrow F[Q] = (w^1, w^2, \dots, w^M)$
- N very large (database size), M small (query size)



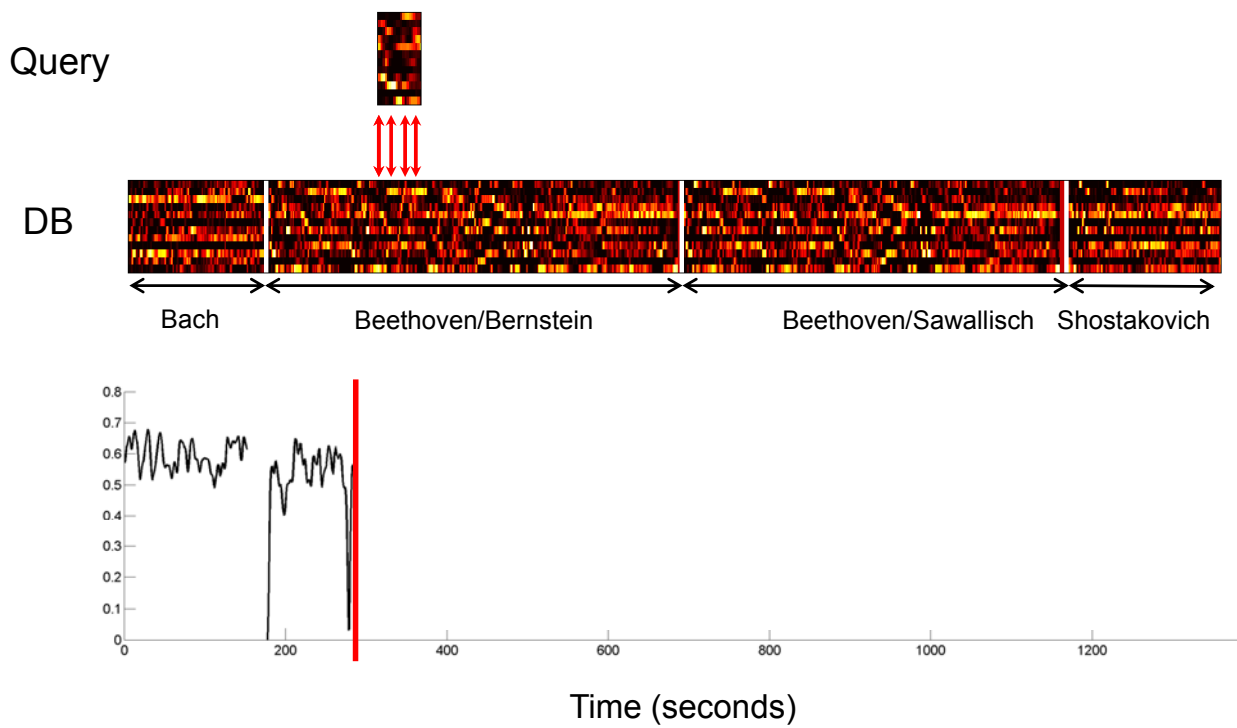
$$\Delta(i) := \text{local distance}((v^i, v^{i+1}, \dots, v^{i+M-1}), (w^1, w^2, \dots, w^M))$$

$$\rightsquigarrow \text{Matching curve } \Delta : [1 : N] \rightarrow [0, 1]$$

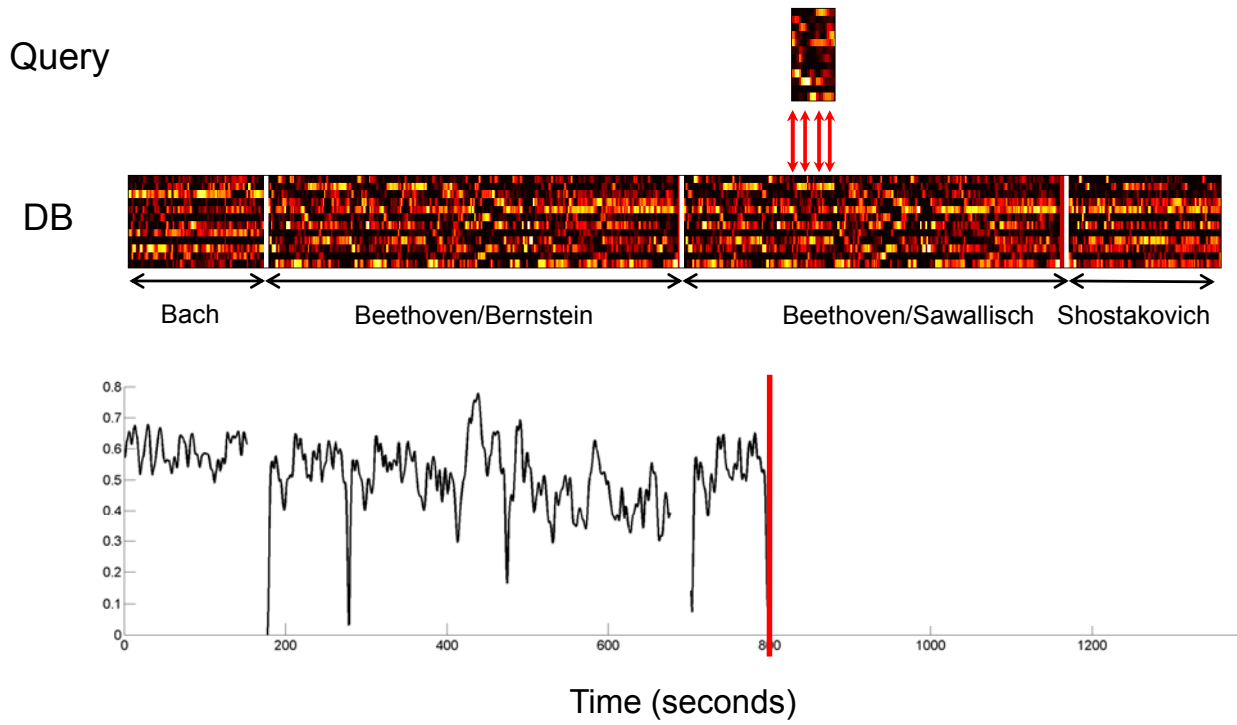
Matching Procedure



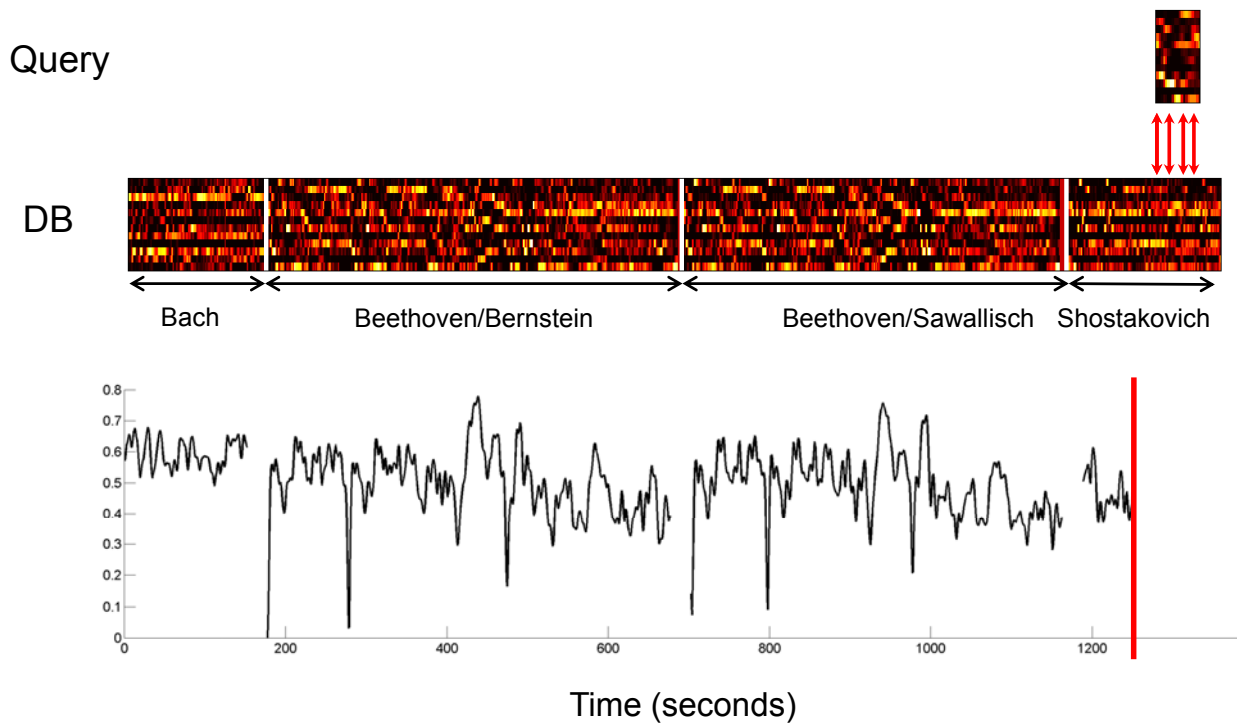
Matching Procedure



Matching Procedure



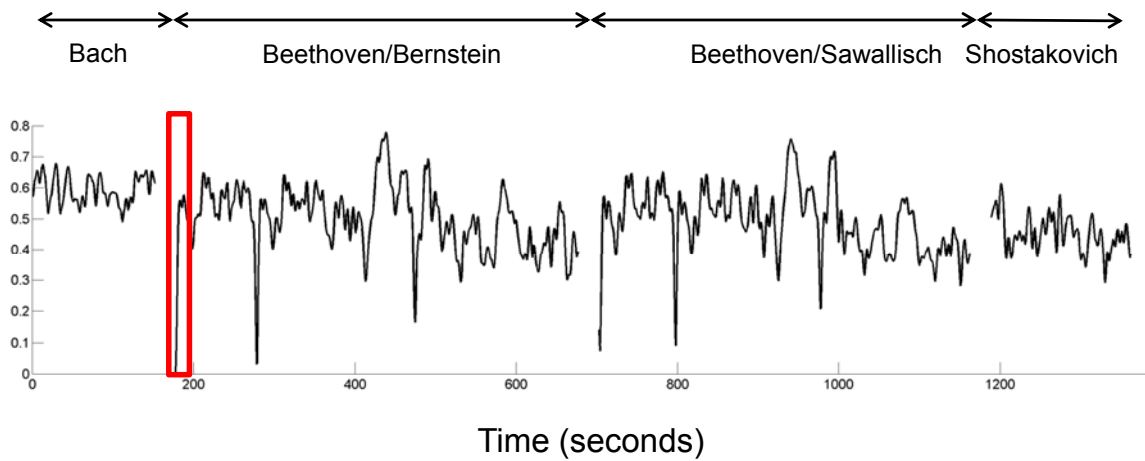
Matching Procedure



Matching Procedure

Matching curve

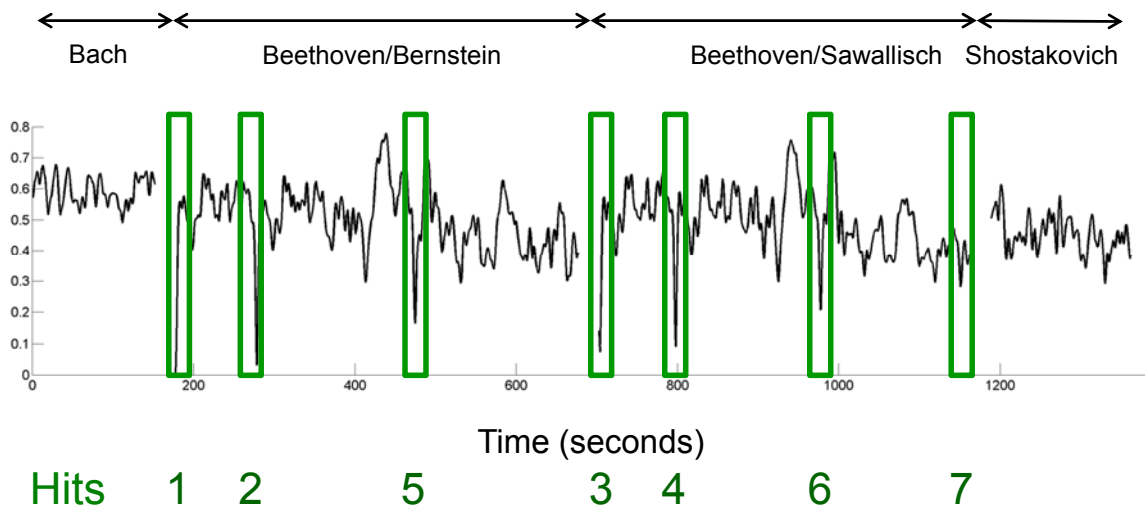
Query: Beethoven's Fifth / Bernstein (first 20 seconds)



Matching Procedure

Matching curve

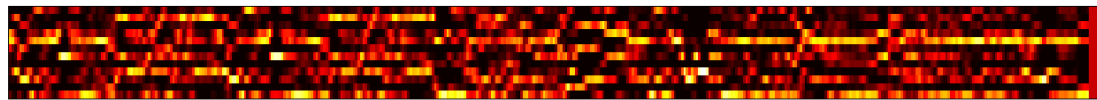
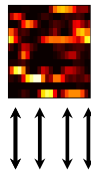
Query: Beethoven's Fifth / Bernstein (first 20 seconds)



Matching Procedure

Problem: How to deal with tempo differences?

Karajan is much faster than Bernstein!



Beethoven/Karajan



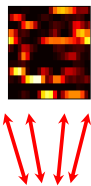
Matching curve does not indicate any hits!

Time (seconds)

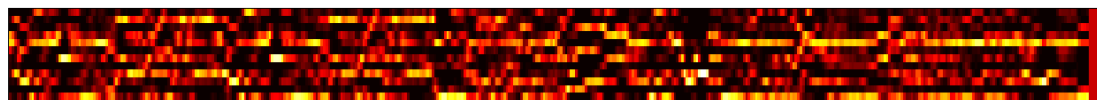
Matching Procedure

1. Strategy: Usage of local warping

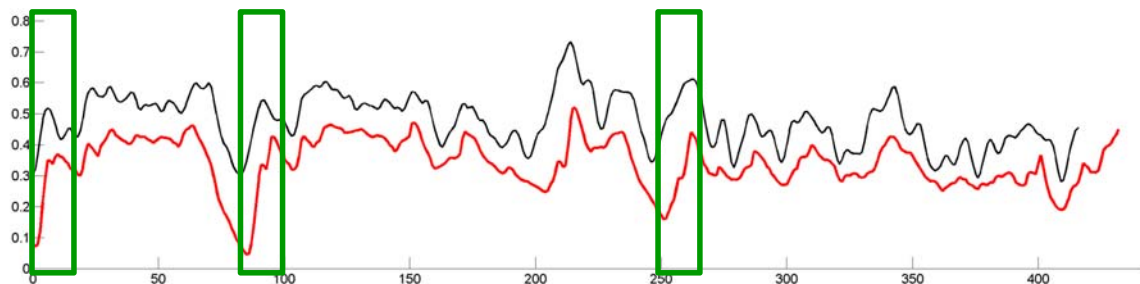
Karajan is much faster than Bernstein!



Warping strategies are computationally expensive and hard for indexing.



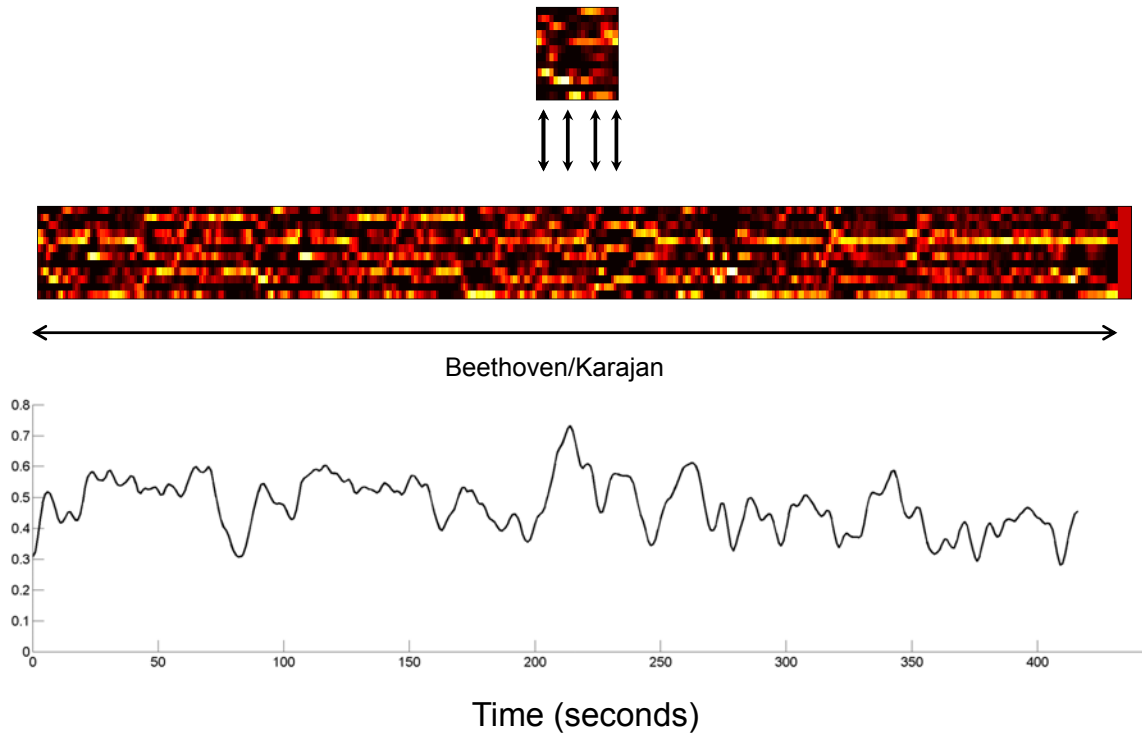
Beethoven/Karajan



Time (seconds)

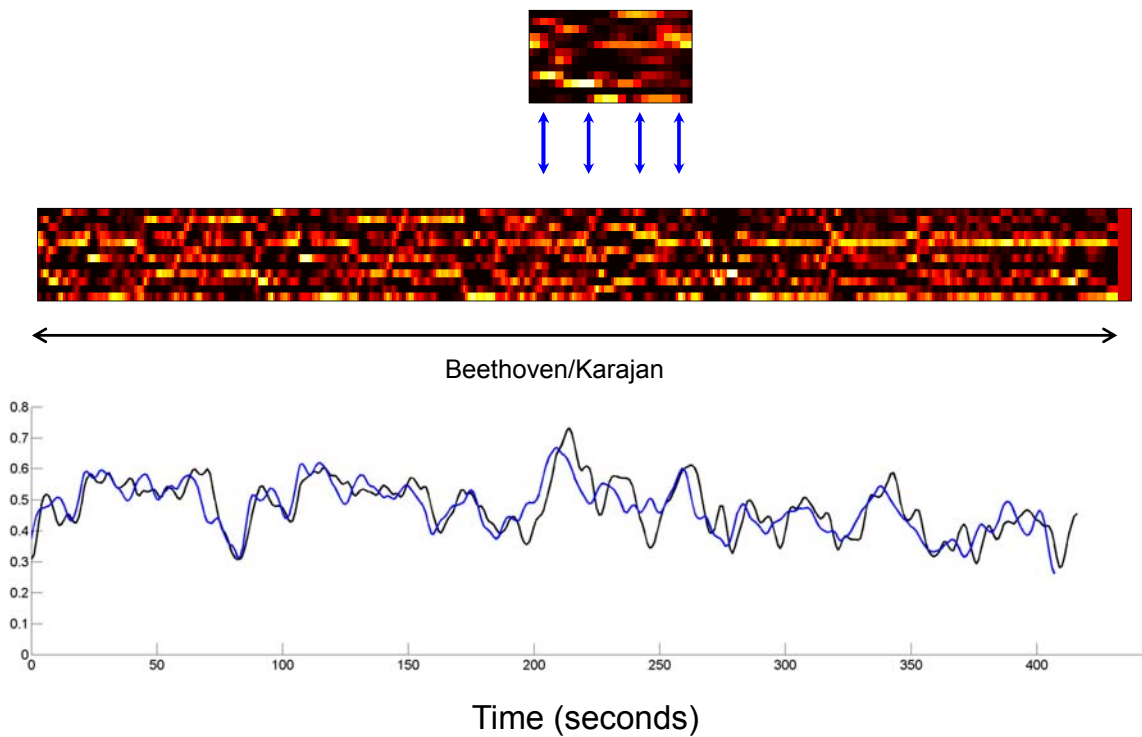
Matching Procedure

2. Strategy: Usage of multiple scaling



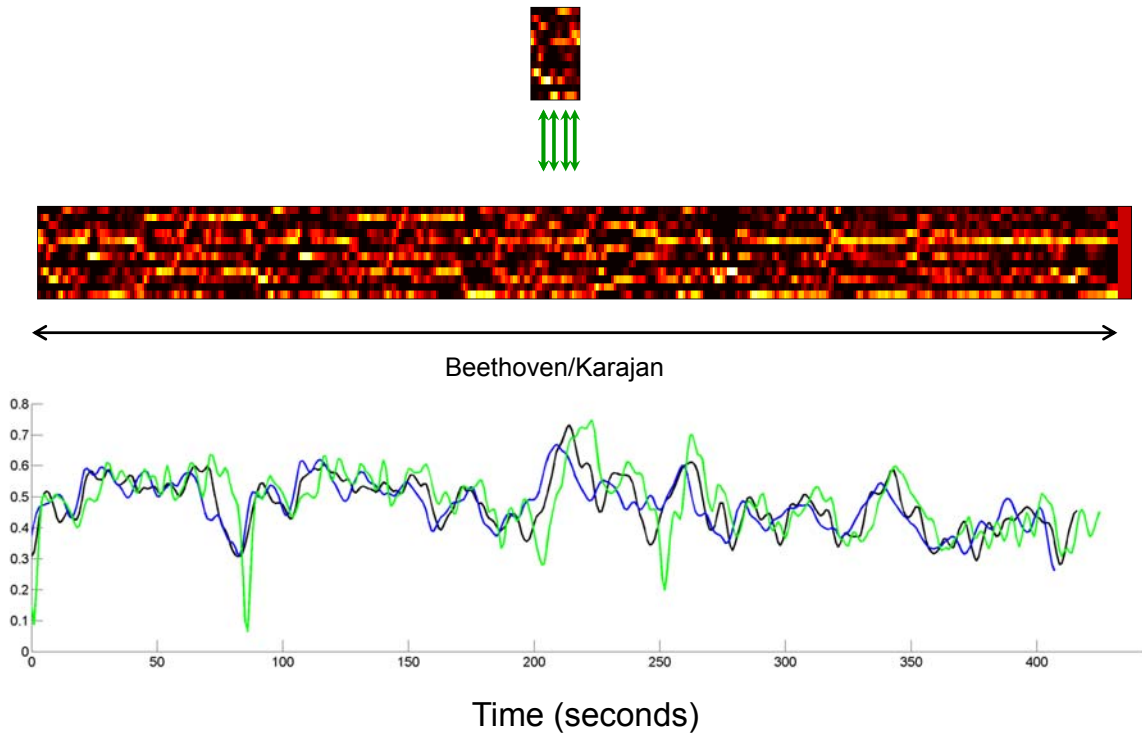
Matching Procedure

2. Strategy: Usage of multiple scaling



Matching Procedure

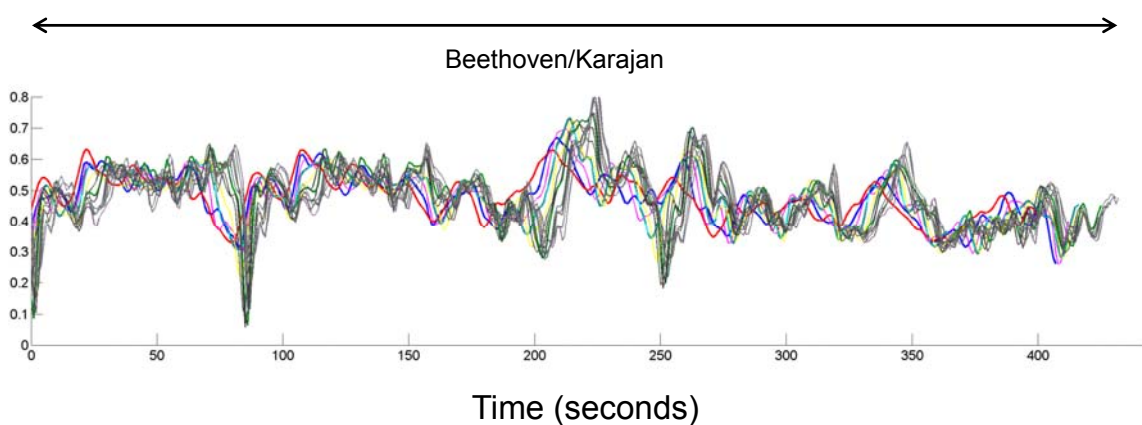
2. Strategy: Usage of multiple scaling



Matching Procedure

2. Strategy: Usage of multiple scaling

Query resampling simulates tempo changes

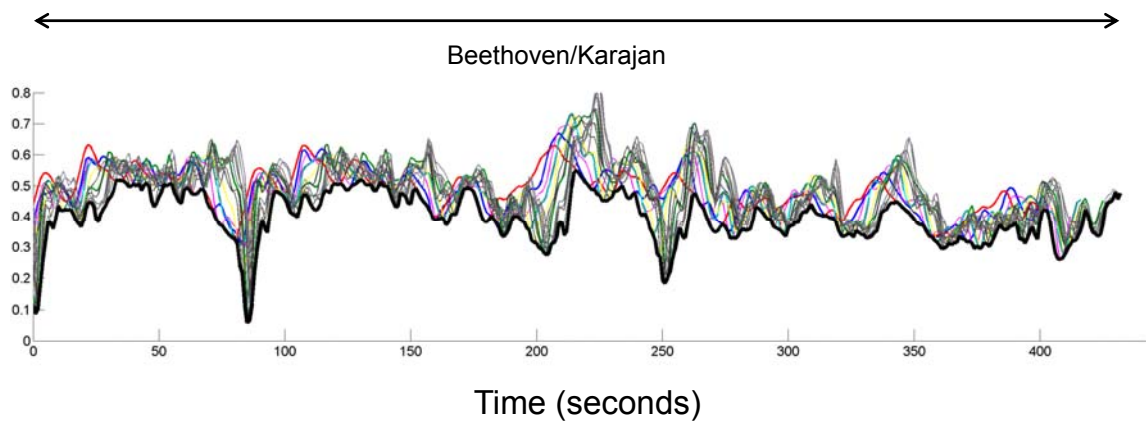


Matching Procedure

2. Strategy: Usage of multiple scaling

Query resampling simulates tempo changes

Minimize over all curves



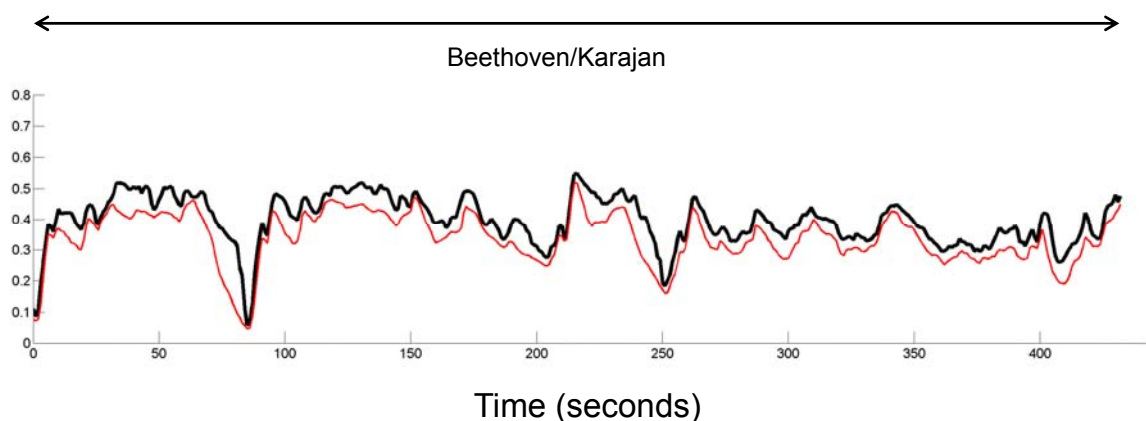
Matching Procedure

2. Strategy: Usage of multiple scaling

Query resampling simulates tempo changes

Minimize over all curves

Resulting curve is similar *warping curve*



Experiments

- Audio database \approx 110 hours, 16.5 GB
- Preprocessing \rightarrow chroma features, 40.3 MB
- Query clip \approx 20 seconds
- Retrieval time \approx 10 seconds (using MATLAB)

Experiments

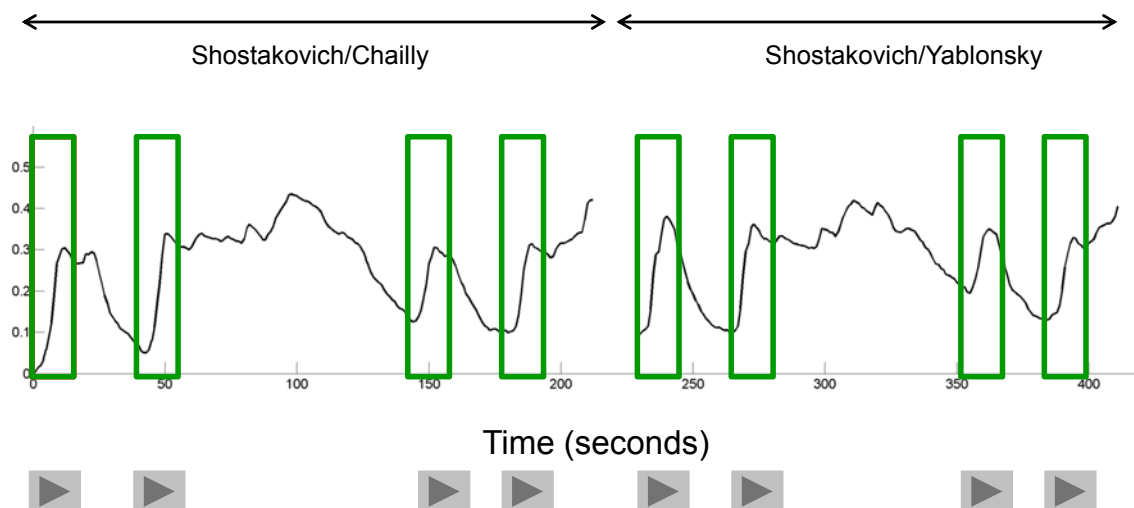
Query: Beethoven's Fifth / Bernstein (first 20 seconds)

| Rank | Piece | Position | |
|------|------------------------------------|-----------|---|
| 1 | Beethoven's Fifth/Bernstein | 0 - 21 | ▶ |
| 2 | Beethoven's Fifth/Bernstein | 101 - 122 | ▶ |
| 3 | Beethoven's Fifth/Karajan | 86 - 103 | ▶ |
| ⋮ | ⋮ | ⋮ | ⋮ |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 10 | Beethoven's Fifth/Karajan | 252 - 271 | ▶ |
| 11 | Beethoven (Liszt) Fifth/Scherbakov | 0 - 19 | ▶ |
| 12 | Beethoven's Fifth/Sawallisch | 275 - 296 | ▶ |
| 13 | Beethoven (Liszt) Fifth/Scherbakov | 86 - 103 | ▶ |
| 14 | Schumann Op. 97,1/Levine | 28 - 43 | ▶ |

Experiments

Query: Shostakovich, Waltz / Chailly (first 21 seconds) ▶

Expected hits



Experiments

Query: Shostakovich, Waltz / Chailly (first 21 seconds) ▶

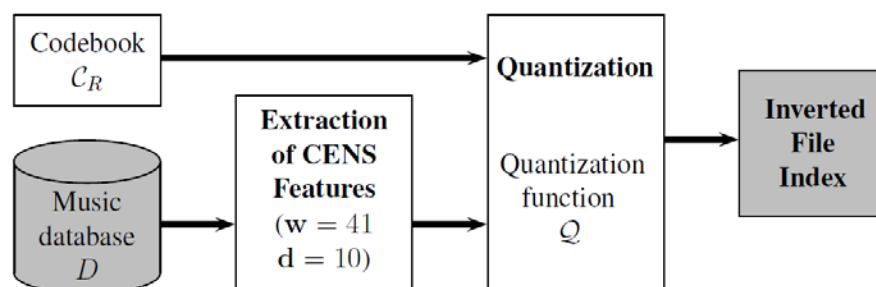
| Rank | Piece | Position | |
|------|------------------------------|-----------|---|
| 1 | Shostakovich/Chailly | 0 - 21 | ▶ |
| 2 | Shostakovich/Chailly | 41 - 60 | ▶ |
| 3 | Shostakovich/Chailly | 180 - 198 | ▶ |
| 4 | Shostakovich/Yablonsky | 1 - 19 | ▶ |
| 5 | Shostakovich/Yablonsky | 36 - 52 | ▶ |
| 6 | Shostakovich/Yablonsky | 156 - 174 | ▶ |
| 7 | Shostakovich/Chailly | 144 - 162 | ▶ |
| 8 | Bach BWV 582/Chorzempa | 358 - 373 | ▶ |
| 9 | Beethoven Op. 37,1/Toscanini | 12 - 28 | ▶ |
| 10 | Beethoven Op. 37,1/Pollini | 202 - 218 | ▶ |

Indexing

- Matching procedure is linear in size of database
- Retrieval time was 10 seconds for 110 hours of audio
 - Much too slow
 - Does not scale to millions of songs
 - Need of indexing methods

Indexing

General procedure



- Convert database into feature sequence (chroma)
- Quantize features with respect to a fixed codebook
- Create an inverted file index
 - contains for each codebook vector an inverted list
 - each list contains feature indices in ascending order

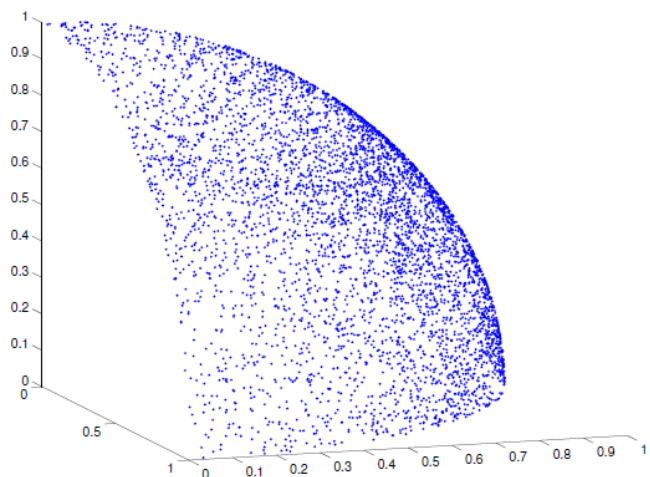
Indexing

Quantization

- Feature space

$$\mathcal{F} := \{v \in [0, 1]^{12} \mid \|v\|_2 = 1\}$$

Visualization (3D)



Indexing

Quantization

- Feature space

$$\mathcal{F} := \{v \in [0, 1]^{12} \mid \|v\|_2 = 1\}$$

- Codebook selection
of suitable size R

$$\{c_1, \dots, c_R\} \subset \mathcal{F}$$

- Quantization using nearest neighbors

$$Q[v] := \operatorname{argmin}_{r \in [1:R]} \arccos(\langle v, c_r \rangle)$$

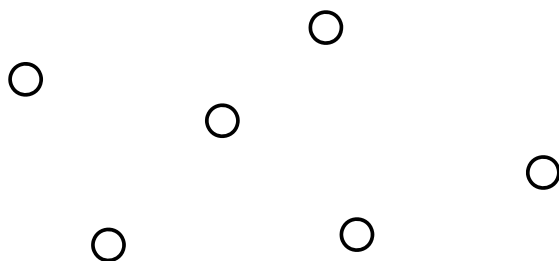
Indexing

How to derive a good codebook?

- Codebook selection by unsupervised learning
 - Linde–Buzo–Gray (LBG) algorithm
 - similar to k-means
 - adjust algorithm to spheres
- Codebook selection based on musical knowledge

Indexing

LBG algorithm

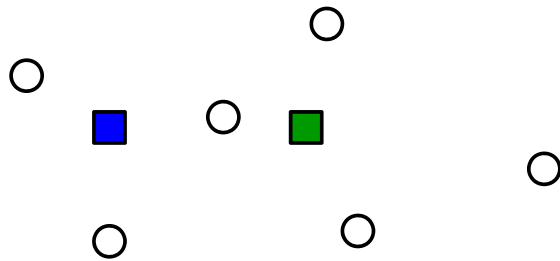


Steps:

1. Initialization of codebook vectors
2. Assignment
3. Recalculation
4. Iteration (back to 2.)

Indexing

LBG algorithm

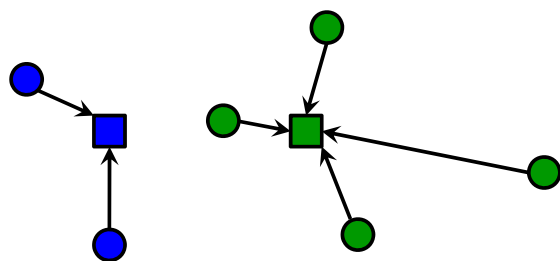


Steps:

1. Initialization of codebook vectors
2. Assignment
3. Recalculation
4. Iteration (back to 2.)

Indexing

LBG algorithm

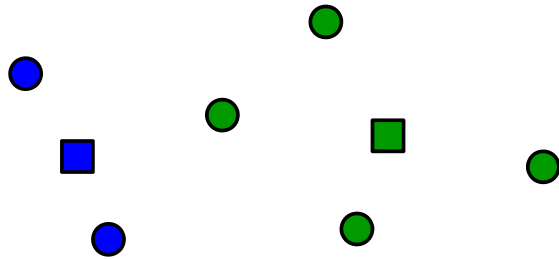


Steps:

1. Initialization of codebook vectors
2. Assignment
3. Recalculation
4. Iteration (back to 2.)

Indexing

LBG algorithm

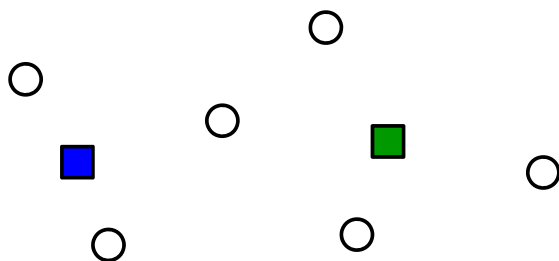


Steps:

1. Initialization of codebook vectors
2. Assignment
3. **Recalculation**
4. Iteration (back to 2.)

Indexing

LBG algorithm

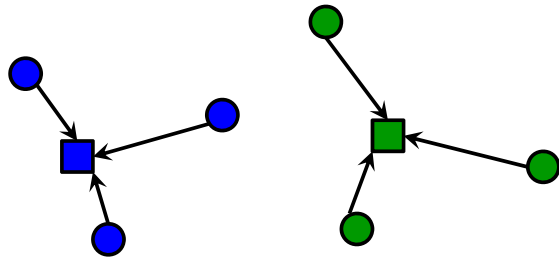


Steps:

1. Initialization of codebook vectors
2. Assignment
3. Recalculation
4. **Iteration (back to 2.)**

Indexing

LBG algorithm

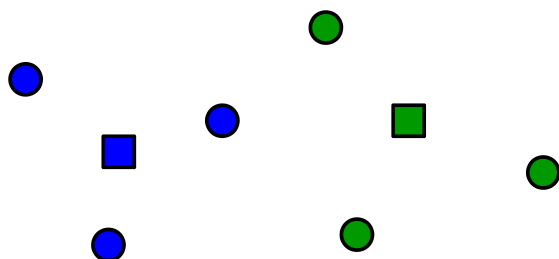


Steps:

1. Initialization of codebook vectors
2. **Assignment**
3. Recalculation
4. Iteration (back to 2.)

Indexing

LBG algorithm

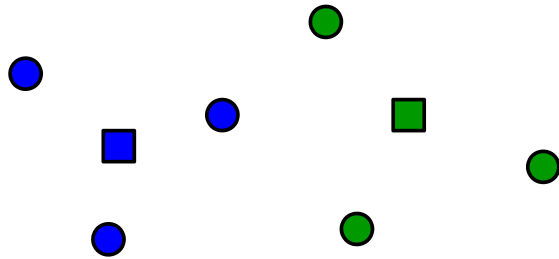


Steps:

1. Initialization of codebook vectors
2. Assignment
3. **Recalculation**
4. Iteration (back to 2.)

Indexing

LBG algorithm



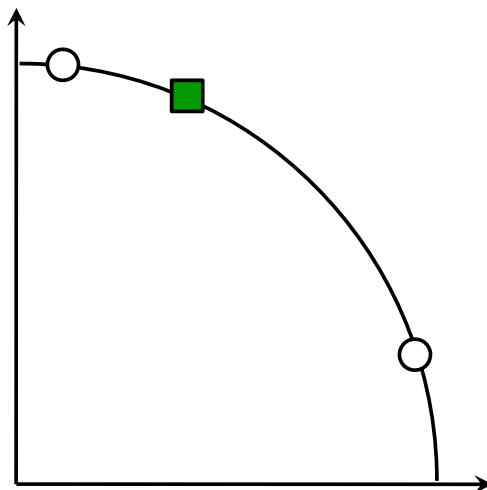
Steps:

1. Initialization of codebook vectors
2. Assignment
3. Recalculation
4. Iteration (back to 2.)

Until convergence

Indexing

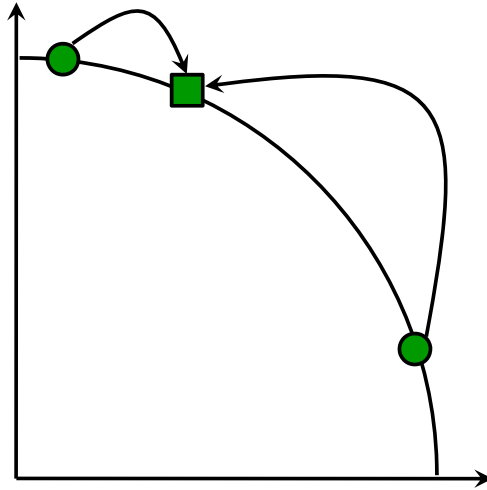
LBG algorithm for spheres



- **Example: 2D**
- Assignment
- Recalculation
- Projection

Indexing

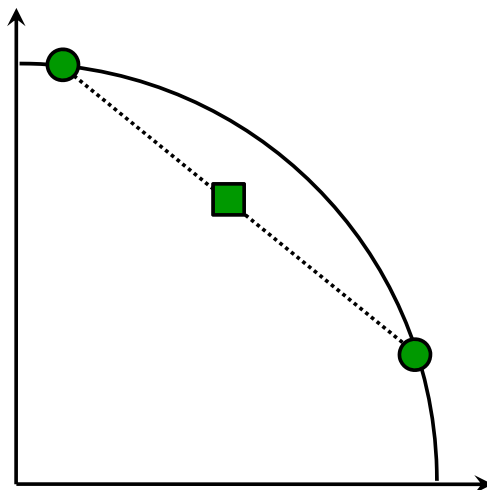
LBG algorithm for spheres



- Example: 2D
- **Assignment**
- Recalculation
- Projection

Indexing

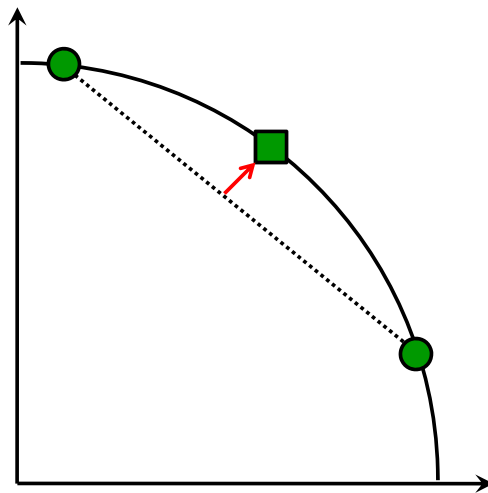
LBG algorithm for spheres



- Example: 2D
- Assignment
- **Recalculation**
- Projection

Indexing

LBG algorithm for spheres



- Example: 2D
- Assignment
- Recalculation
- **Projection**

Indexing

Codebook using musical knowledge

- Observation: Chroma features capture harmonic information
- Example: C-Major $\frac{1}{\sqrt{3}}(1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0)$
- Example: C#-Major $\frac{1}{\sqrt{3}}(0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0)$
- Experiments: For more than 95% of all chroma features >50% of energy lies in at most 4 components

Indexing

Codebook using musical knowledge

- C-Major $\frac{1}{\sqrt{3}}(1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0) = \frac{1}{\sqrt{3}}(\delta_1 + \delta_5 + \delta_8)$
- C#-Major $\frac{1}{\sqrt{3}}(0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0) = \frac{1}{\sqrt{3}}(\delta_2 + \delta_6 + \delta_9)$
- Choose codebook to contain n -chords for $n=1,2,3,4$

| n | 1 | 2 | 3 | 4 | |
|----------|------------|---|--|---|-----|
| template | δ_j | $\frac{1}{\sqrt{2}}(\delta_{k_1} + \delta_{k_2})$ | $\frac{1}{\sqrt{3}}(\delta_{r_1} + \delta_{r_2} + \delta_{r_3})$ | $\frac{1}{\sqrt{4}}(\delta_{n_1} + \delta_{n_2} + \delta_{n_3} + \delta_{n_4})$ | |
| # | 12 | 66 | 220 | 495 | 793 |

Indexing

Codebook using musical knowledge

Additional consideration of harmonics in chord templates

Example: 1-chord C

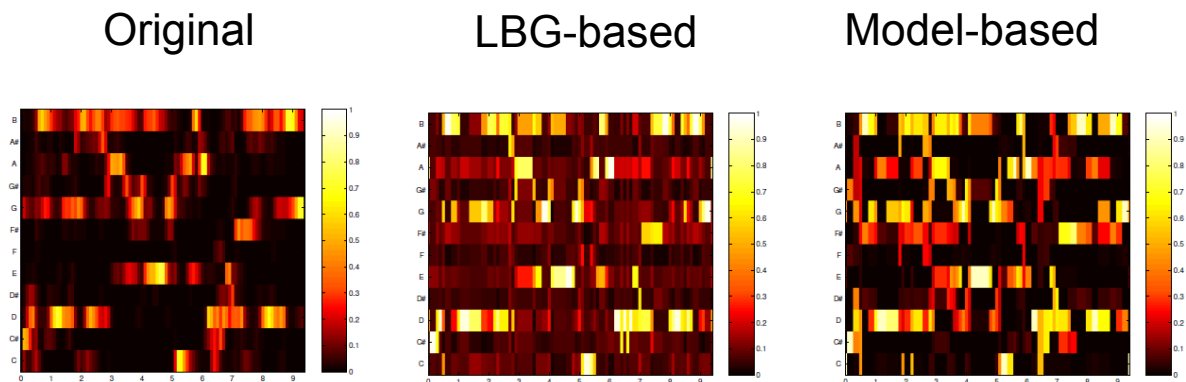
| Harmonics | 1 | 2 | 3 | 4 | 5 | 6 |
|-----------|-----|-----|-----|-----|-----|-----|
| Pitch | C3 | C4 | G4 | C5 | E5 | G5 |
| Frequency | 131 | 262 | 392 | 523 | 654 | 785 |
| Chroma | C | C | G | C | E | C |

Replace δ_1 by $w_1\delta_1 + w_2\delta_1 + w_3\delta_8 + w_4\delta_1 + w_5\delta_5 + w_6\delta_8$
with suitable weights for the harmonics

Indexing

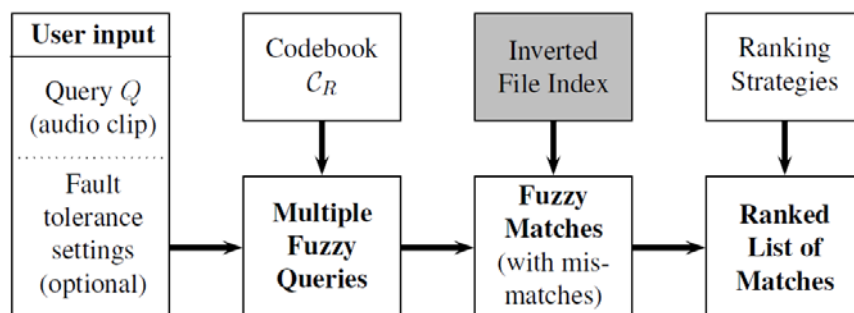
Quantization

Original chromagram and projections on codebooks



Indexing

Query and retrieval stage



- Query consists of a short audio clip (10-40 seconds)
- Specification of fault tolerance setting
 - fuzziness of query
 - number of admissible mismatches
 - tolerance to tempo variations
 - tolerance to modulations

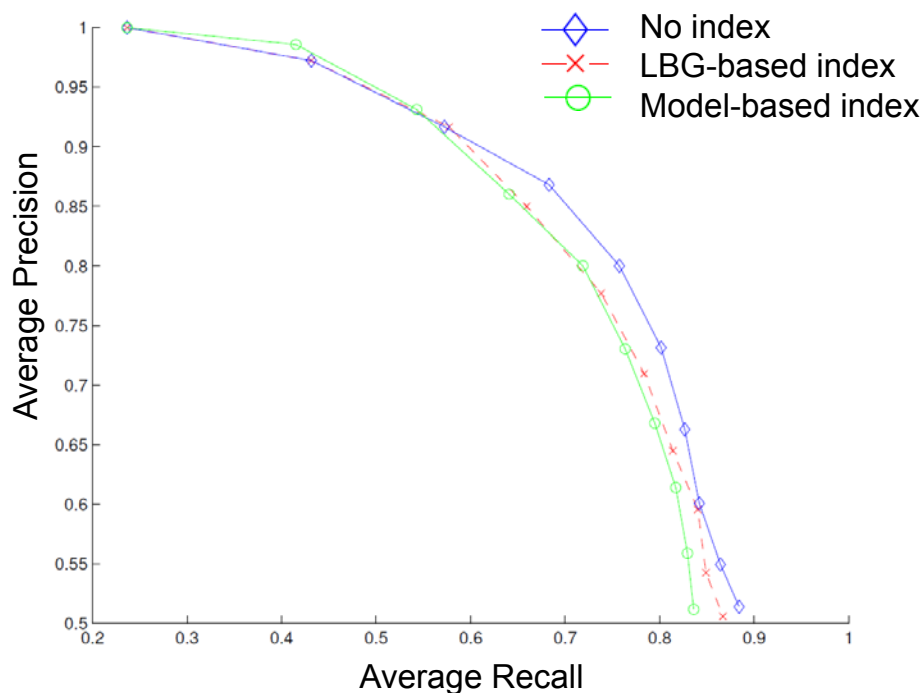
Indexing

Retrieval results

- Medium sized database
 - 500 pieces
 - 112 hours of audio
 - mostly classical music
- Selection of various queries
 - 36 queries
 - duration between 10 and 40 seconds
 - hand-labelled matches in database
- Indexing leads to speed-up factor between 15 and 20 (depending on query length)
- Only small degradation in precision and recall

Indexing

Retrieval results



Indexing

Conclusions

- Described method suitable for medium-sized databases
 - index is assumed to be in main memory
 - inverted lists may be long
- Goal was to find **all** meaningful matches
 - high-degree of fault-tolerance required (fuzzyness, mismatches)
 - number of intersections and unions may explode
- What to do when dealing with millions of songs?
- Can the quantization be avoided?
- Better indexing and retrieval methods needed!
 - kd-trees
 - locality sensitive hashing
 - ...

Conclusions (Audio Matching)

Matching procedure

- Strategy: Exact matching and multiple scaled queries
 - simulate tempo variations by feature resampling
 - different queries correspond to different tempi
 - indexing possible
- Strategy: Dynamic time warping
 - subsequence variant
 - more flexible (in particular for longer queries)
 - indexing hard

Conclusions (Audio Matching)

Audio Features

Strategy: Absorb variations already at feature level

- Chroma → invariance to timbre
- Normalization → invariance to dynamics
- Smoothing → invariance to local time deviations

**Message: There is no standard chroma feature!
Variants can make a huge difference!**

Feature Design

- Enhancement of chroma features
- Usage of audio matching framework for evaluating the quality of obtained audio features
- Usage of matching curves as mid-level representation to reveal a feature's robustness and discriminative capability

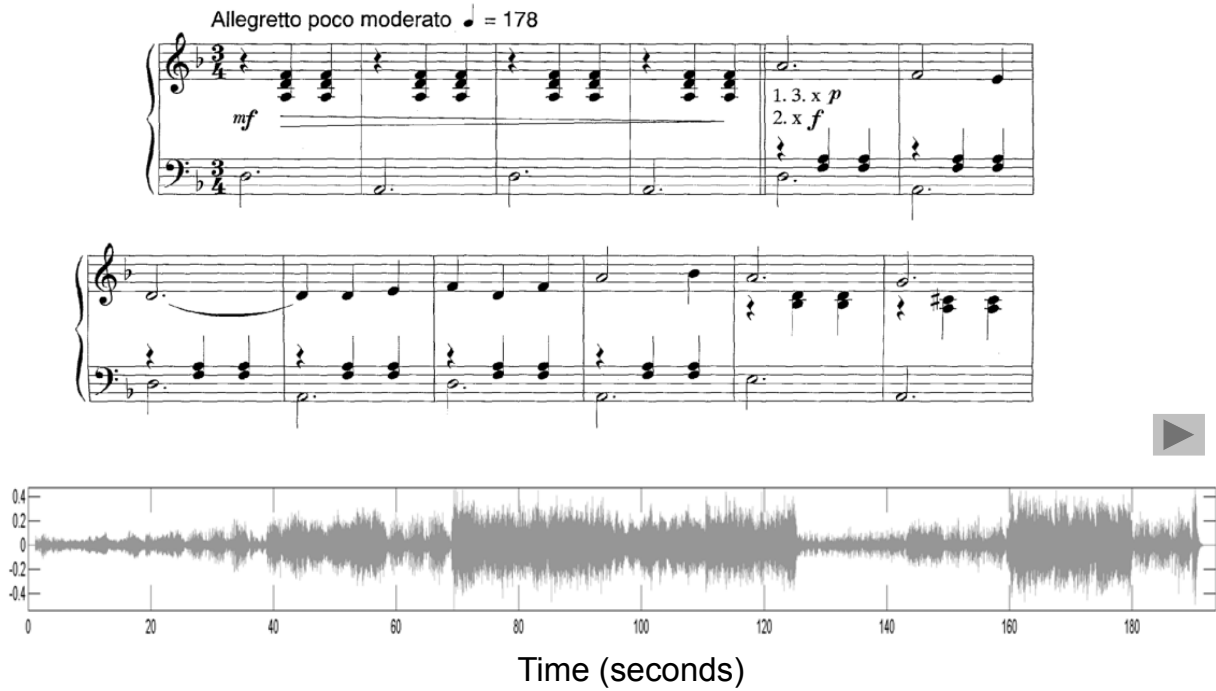
M. Müller and S. Ewert (2010):

Towards Timbre-Invariant Audio Features for Harmony-Based Music.

IEEE Trans. on Audio, Speech & Language Processing, Vol. 18, No. 3,
pp. 649-662.

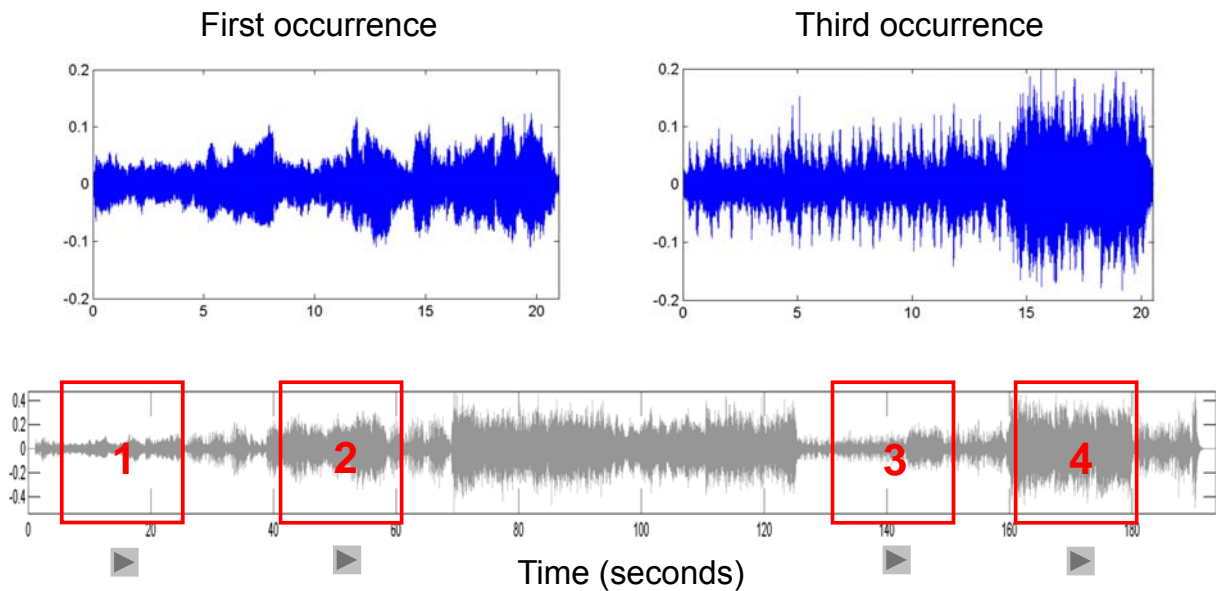
[Müller/Ewert, IEEE-TASLP 2010]

Motivation: Audio Matching

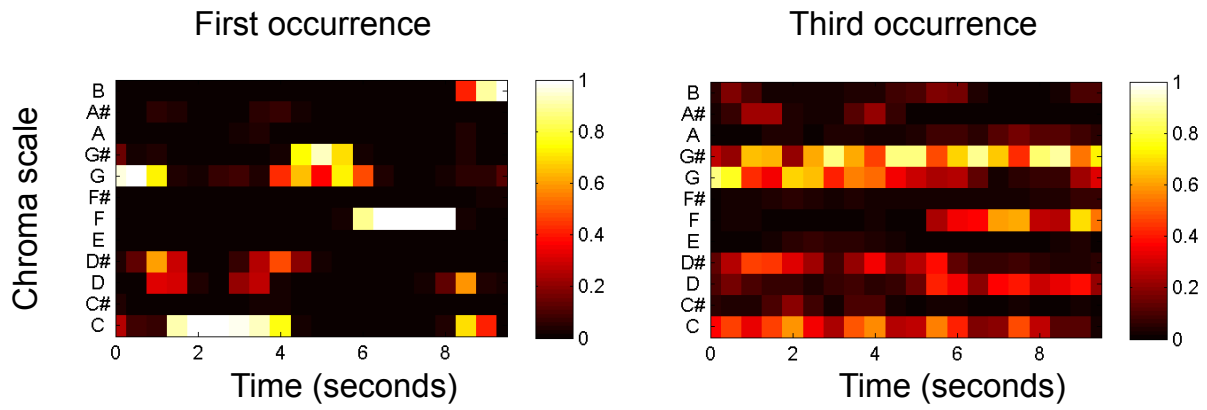


Motivation: Audio Matching

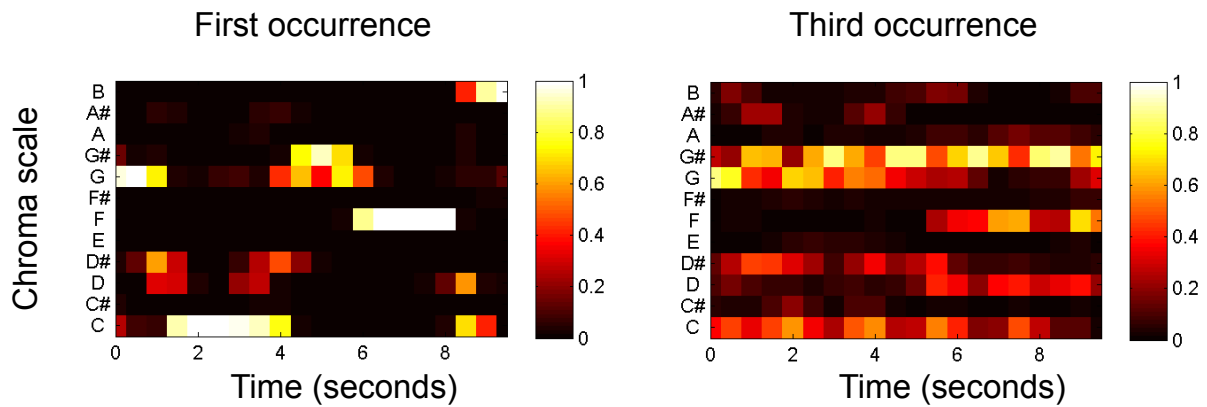
Four occurrences of the main theme



Chroma Features

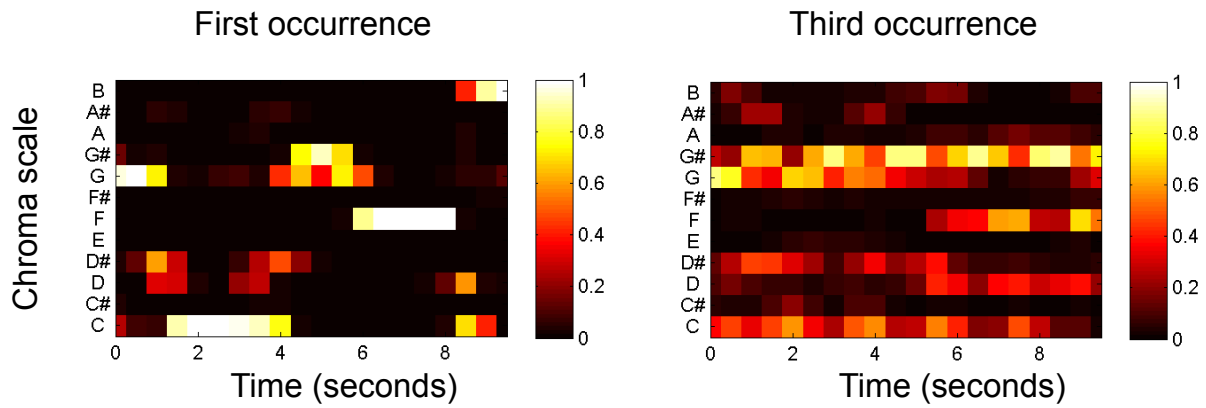


Chroma Features



How to make chroma features more robust to timbre changes?

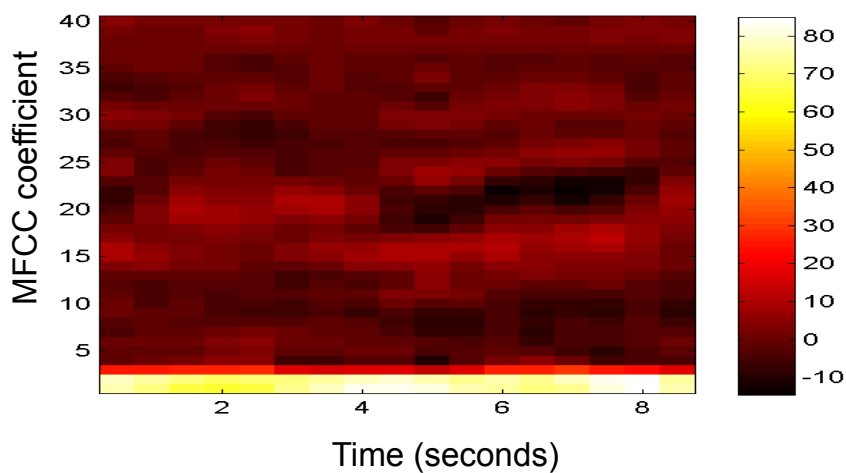
Chroma Features



How to make chroma features more robust to timbre changes?
Idea: Discard timbre-related information

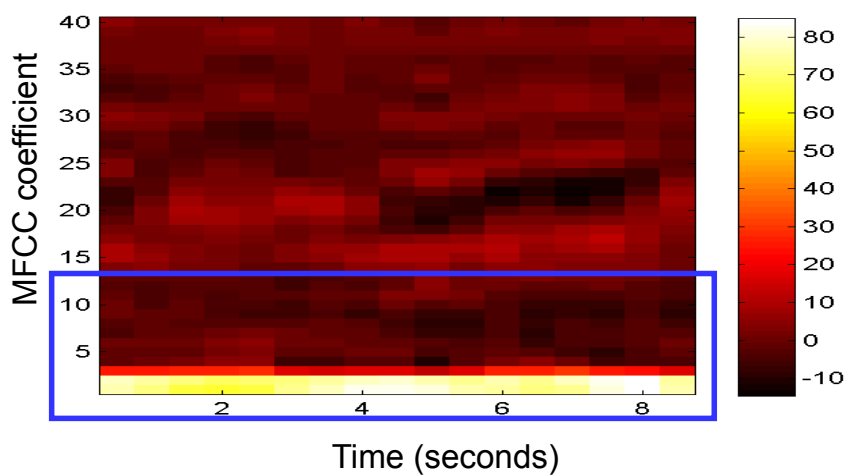
[Müller/Ewert, IEEE-TASLP 2010]

MFCC Features and Timbre



[Müller/Ewert, IEEE-TASLP 2010]

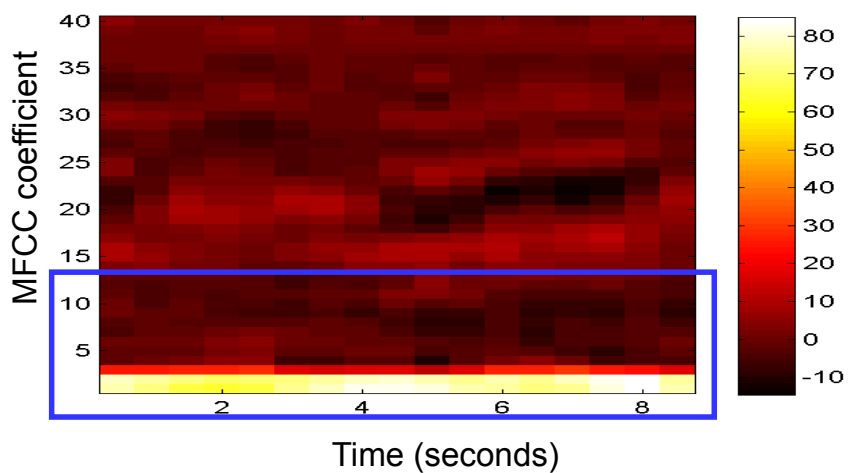
MFCC Features and Timbre



Lower MFCCs \longleftrightarrow Timbre

[Müller/Ewert, IEEE-TASLP 2010]

MFCC Features and Timbre

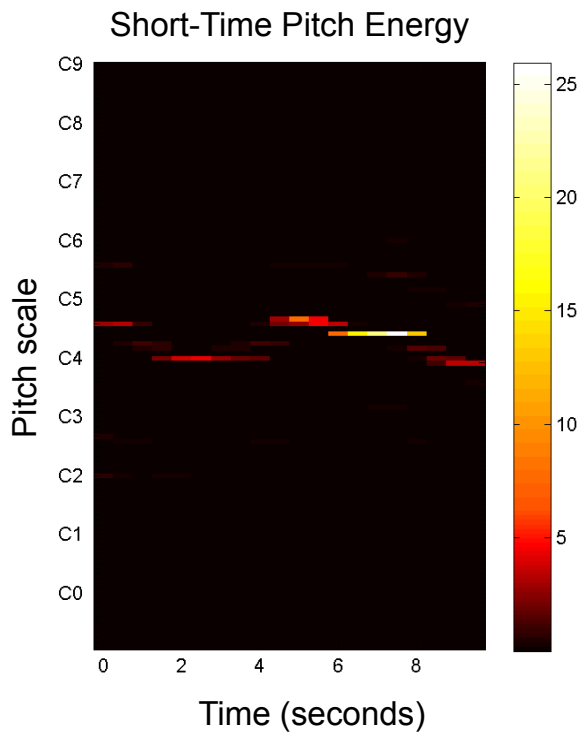


Lower MFCCs \longleftrightarrow Timbre

Idea: Discard lower MFCCs to achieve timbre invariance

[Müller/Ewert, IEEE-TASLP 2010]

Enhancing Timbre Invariance

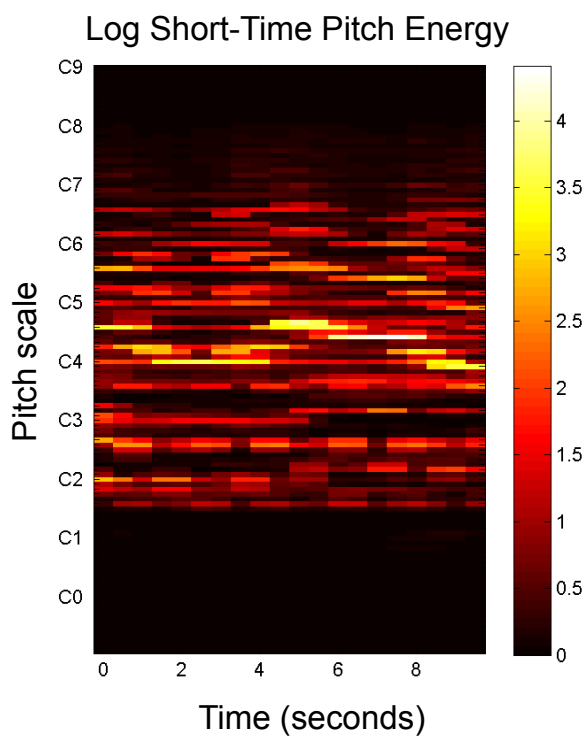


Steps:

1. Log-frequency spectrogram

[Müller/Ewert, IEEE-TASLP 2010]

Enhancing Timbre Invariance

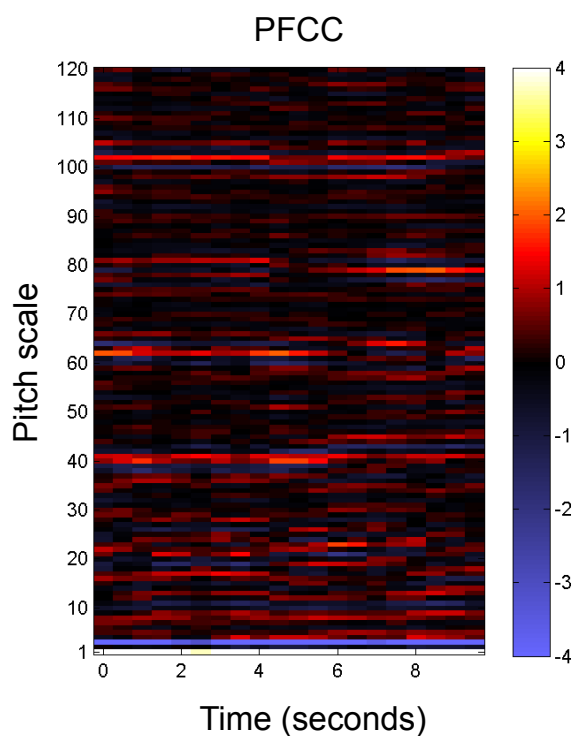


Steps:

1. Log-frequency spectrogram
2. Log (amplitude)

[Müller/Ewert, IEEE-TASLP 2010]

Enhancing Timbre Invariance

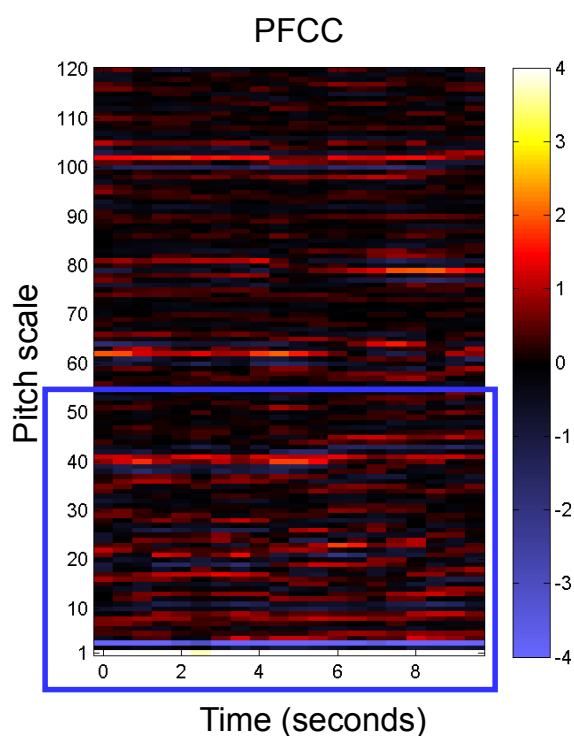


Steps:

1. Log-frequency spectrogram
2. Log (amplitude)
3. DCT

[Müller/Ewert, IEEE-TASLP 2010]

Enhancing Timbre Invariance

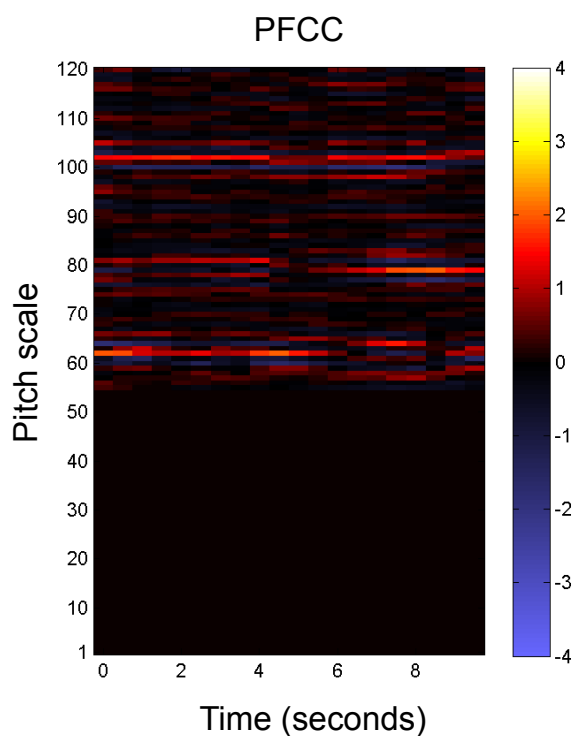


Steps:

1. Log-frequency spectrogram
2. Log (amplitude)
3. DCT
4. Discard lower coefficients [1:n-1]

[Müller/Ewert, IEEE-TASLP 2010]

Enhancing Timbre Invariance

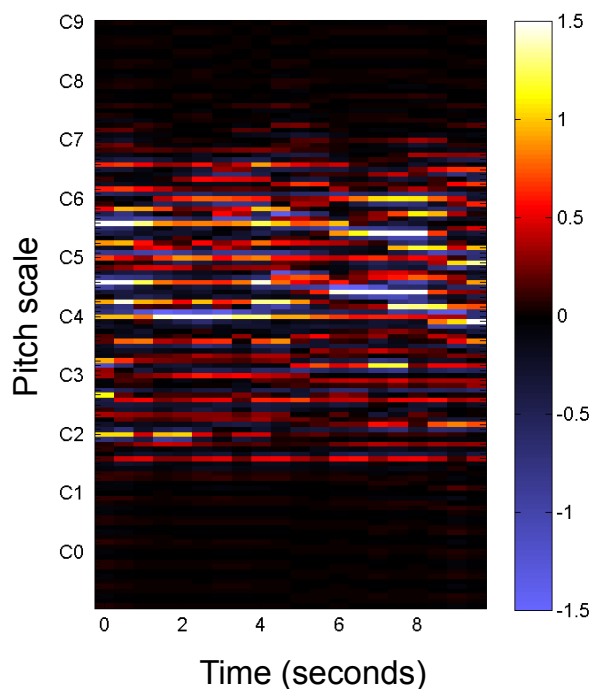


Steps:

1. Log-frequency spectrogram
2. Log (amplitude)
3. DCT
4. Keep upper coefficients [n:120]

[Müller/Ewert, IEEE-TASLP 2010]

Enhancing Timbre Invariance

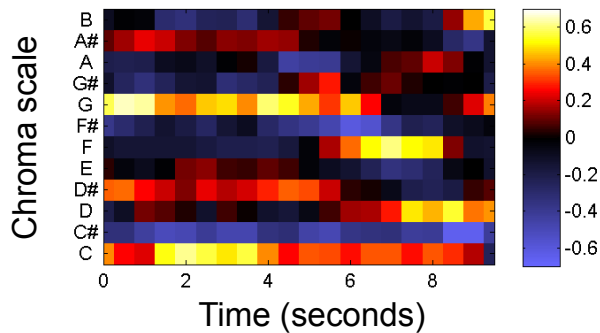


Steps:

1. Log-frequency spectrogram
2. Log (amplitude)
3. DCT
4. Keep upper coefficients [n:120]
5. Inverse DCT

[Müller/Ewert, IEEE-TASLP 2010]

Enhancing Timbre Invariance

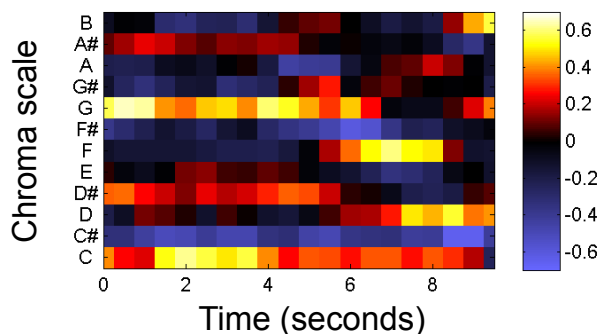


Steps:

1. Log-frequency spectrogram
2. Log (amplitude)
3. DCT
4. Keep upper coefficients [n:120]
5. Inverse DCT
6. Chroma & Normalization

[Müller/Ewert, IEEE-TASLP 2010]

Enhancing Timbre Invariance



Steps:

1. Log-frequency spectrogram
2. Log (amplitude)
3. DCT
4. Keep upper coefficients [n:120]
5. Inverse DCT
6. Chroma & Normalization

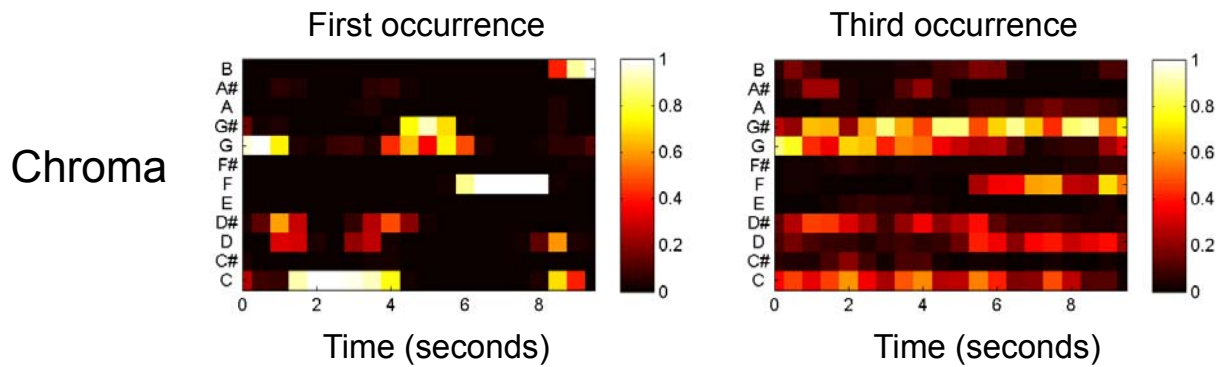
CRP(n)

Chroma DCT-Reduced Log-Pitch

[Müller/Ewert, IEEE-TASLP 2010]

Chroma versus CRP

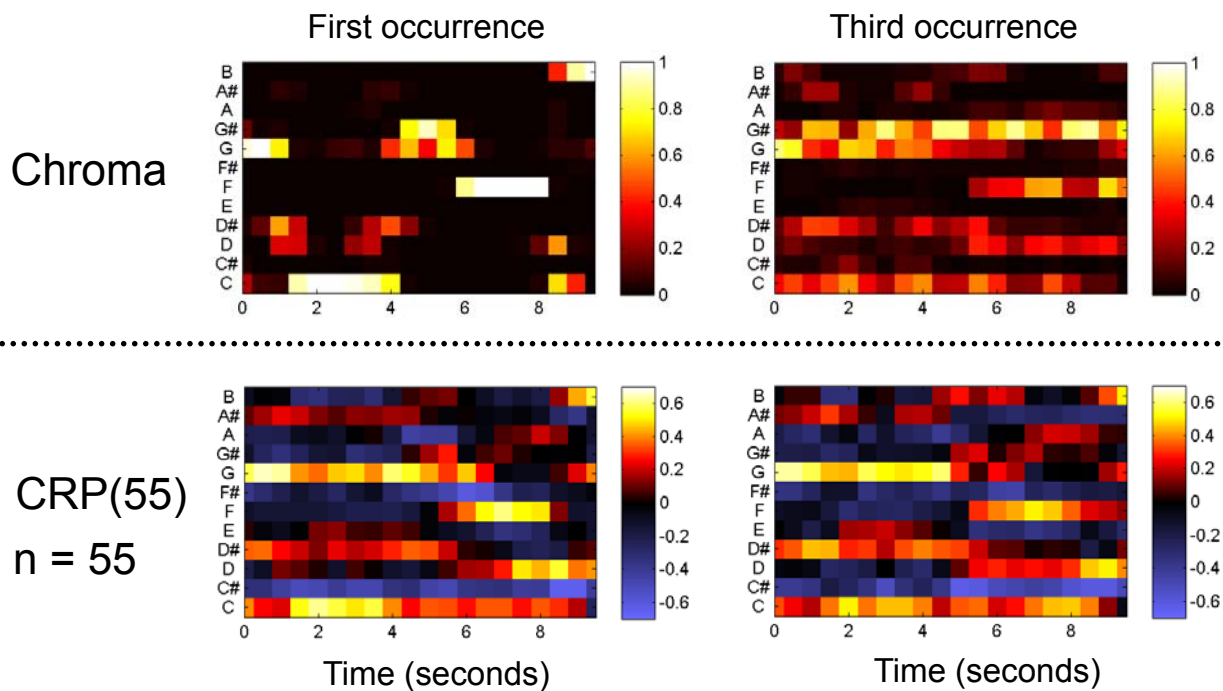
Shostakovich Waltz



[Müller/Ewert, IEEE-TASLP 2010]


Chroma versus CRP

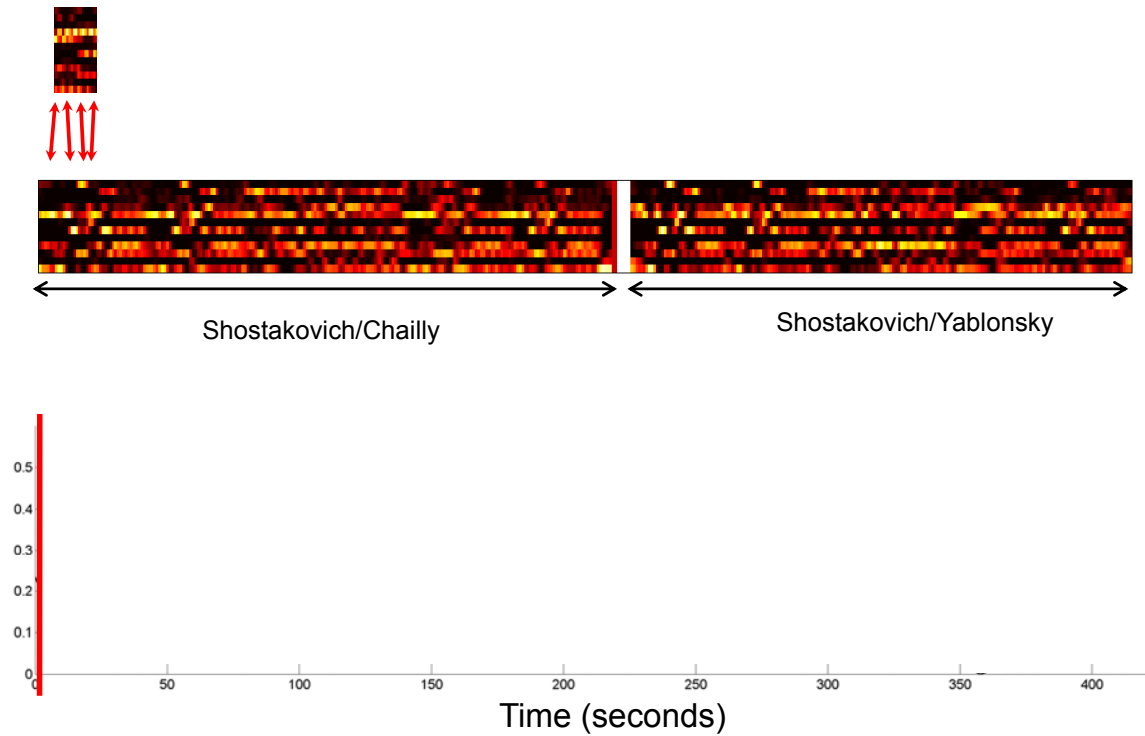
Shostakovich Waltz



[Müller/Ewert, IEEE-TASLP 2010]

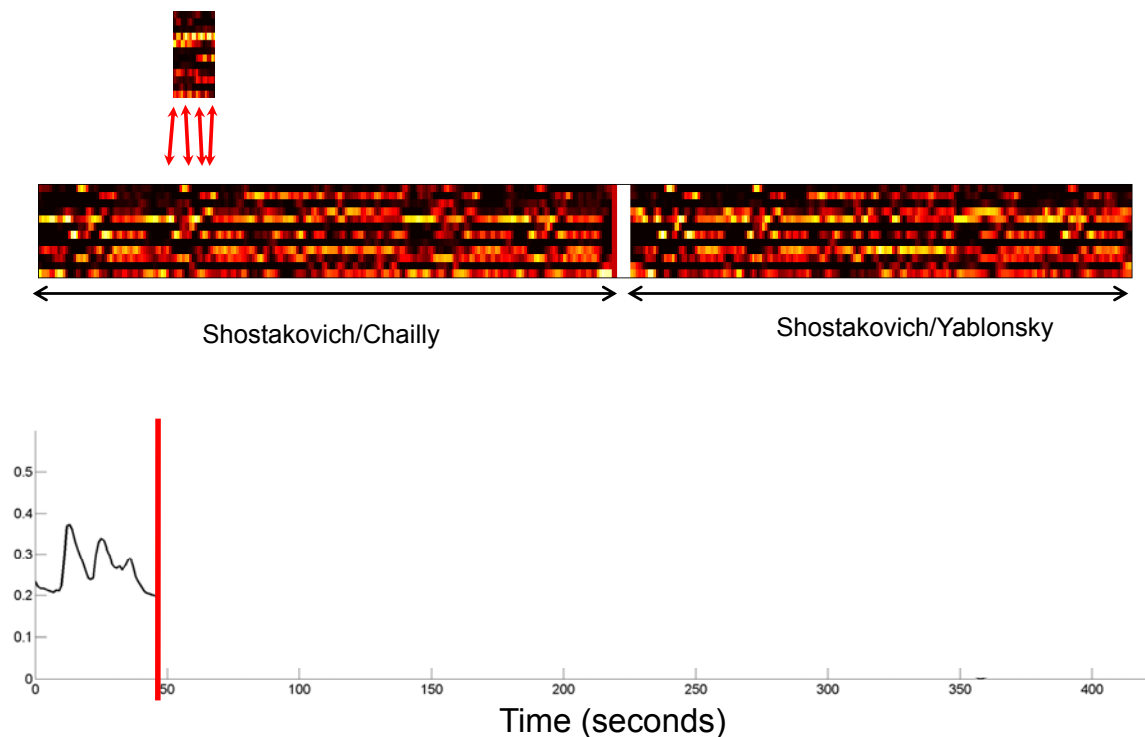
Quality: Audio Matching

Query: Shostakovich, Waltz / Yablonsky (3. occurrence) 




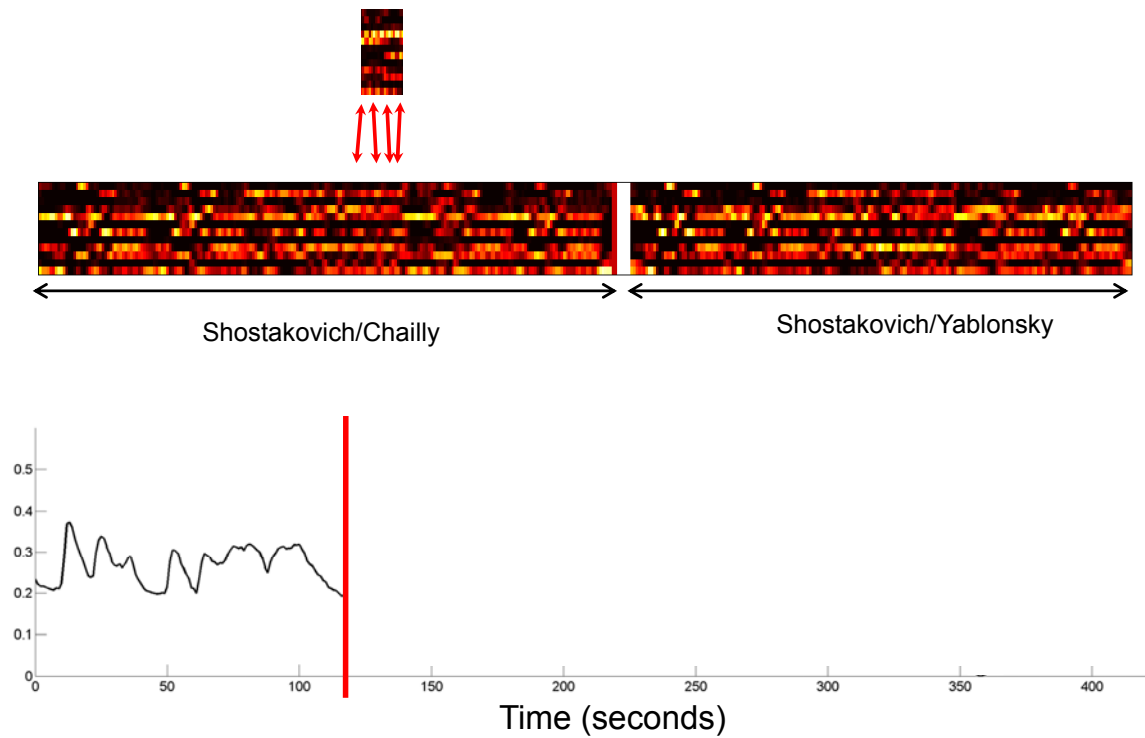
Quality: Audio Matching

Query: Shostakovich, Waltz / Yablonsky (3. occurrence) 



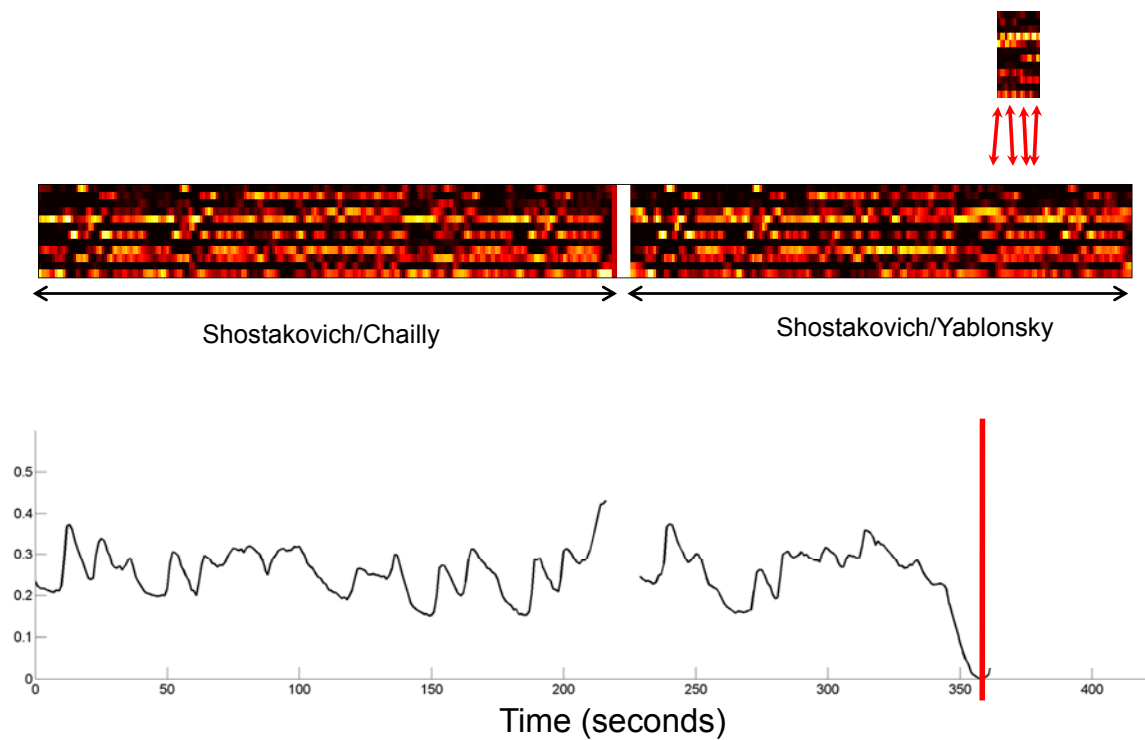
Quality: Audio Matching

Query: Shostakovich, Waltz / Yablonsky (3. occurrence) 



Quality: Audio Matching

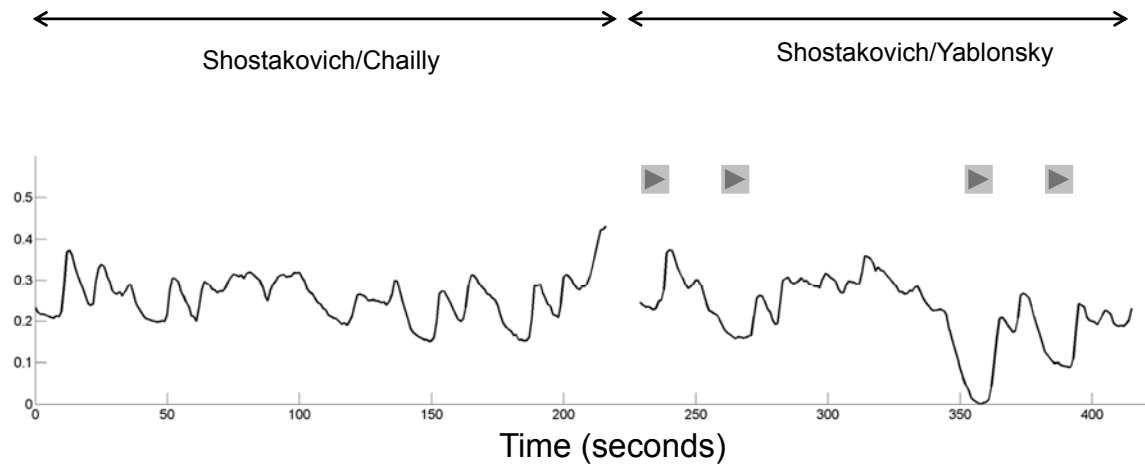
Query: Shostakovich, Waltz / Yablonsky (3. occurrence) 



Quality: Audio Matching

Query: Shostakovich, Waltz / Yablonsky (3. occurrence) ▶

— Standard Chroma (Chroma Pitch)

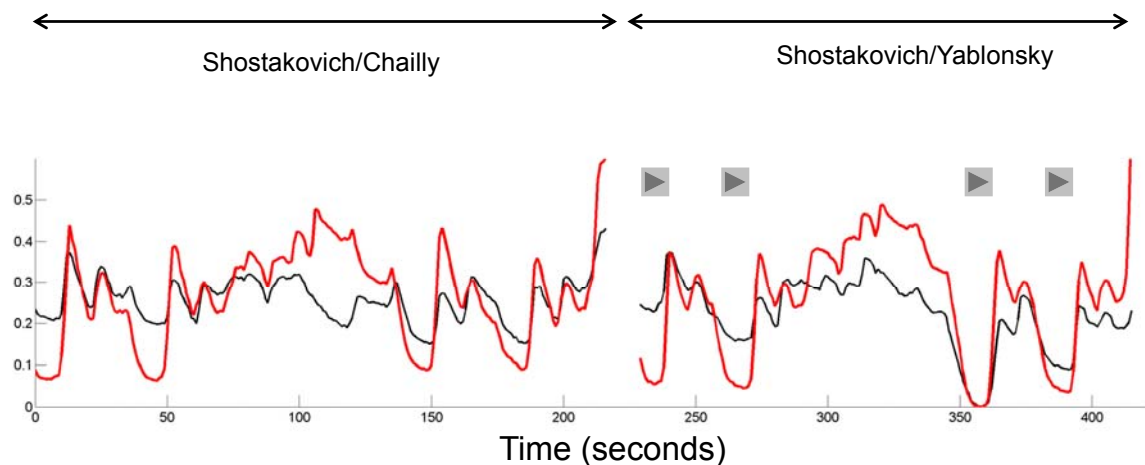


Quality: Audio Matching

Query: Shostakovich, Waltz / Yablonsky (3. occurrence) ▶

— Standard Chroma (Chroma Pitch)

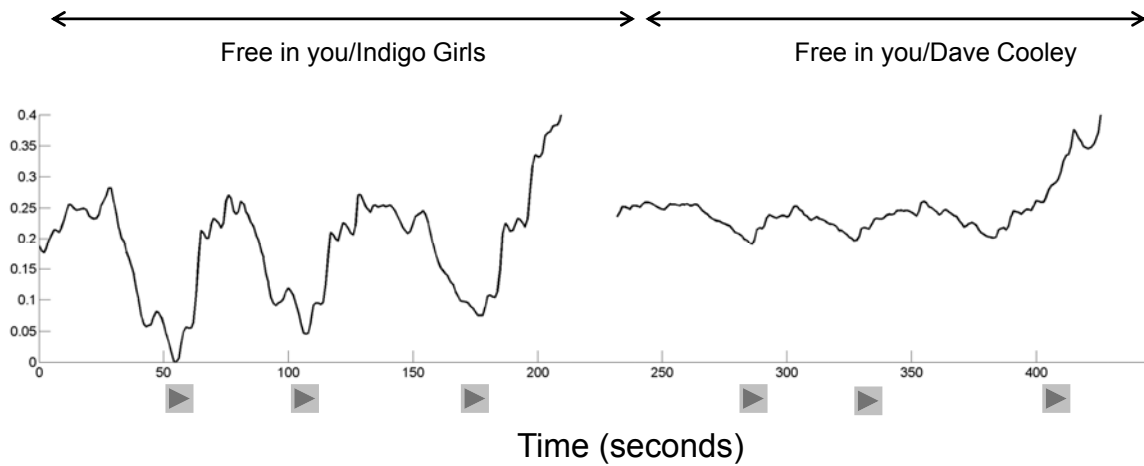
— CRP(55)



Quality: Audio Matching

Query: Free in you / Indigo Girls (1. occurrence) 

— Standard Chroma (Chroma Pitch)

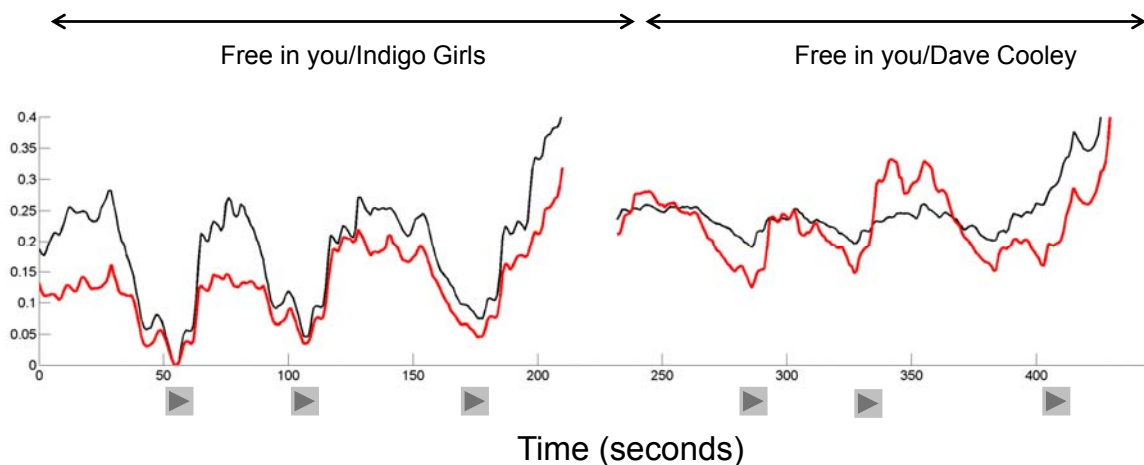


Quality: Audio Matching

Query: Free in you / Indigo Girls (1. occurrence) 

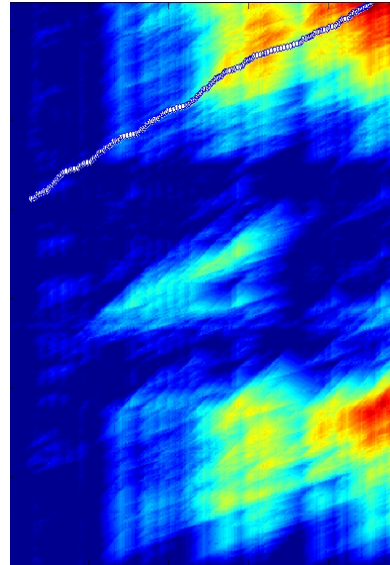
— Standard Chroma (Chroma Pitch)

— CRP(55)



Overview (Audio Retrieval)

- Audio identification (audio fingerprinting)
- Audio matching
- **Cover song identification**



Cover Song Identification

- Gómez/Herrera (ISMIR 2006)
- Casey/Slaney (ISMIR 2006)
- Serrà (ISMIR 2007)
- Ellis/Polioner (ICASSP 2007)
- Serrà/Gómez/Herrera/Serra (IEEE TASLP 2008)

Cover Song Identification

Goal: Given a music recording of a song or piece of music, find all corresponding music recordings within a huge collection that can be regarded as a kind of version, interpretation, or cover song.

- Live versions
- Versions adapted to particular country/region/language
- Contemporary versions of an old song
- Radically different interpretations of a musical piece
- ...

Instance of document-based retrieval!

Cover Song Identification

Motivation

- Automated organization of music collections
“Find me all covers of ...”
- Musical rights management
- Learning about music itself
“Understanding the essence of a song”

Cover Song Identification

Nearly anything can change! But something doesn't change.
Often this is **chord progression** and/or **melody**

| | | | | |
|---|--|-----------------------------|--|---|
| ▶ | Bob Dylan Knockin' on Heaven's Door | key | Avril Lavigne Knockin' on Heaven's Door | ▶ |
| ▶ | Metallica Enter Sandman | timbre | Apocalyptica Enter Sandman | ▶ |
| ▶ | Nirvana Poly [Incesticide Album] | tempo | Nirvana Poly [Unplugged] | ▶ |
| ▶ | Black Sabbath Paranoid | lyrics | Cindy & Bert Der Hund Der Baskerville | ▶ |
| ▶ | AC/DC High Voltage | recording conditions | AC/DC High Voltage [live] | ▶ |
| | | song structure | | |

Cover Song Identification

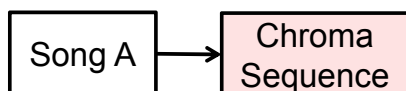
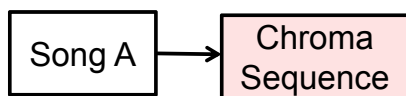
How to compare two different songs?

Song A

Song A

Cover Song Identification

How to compare two different songs?

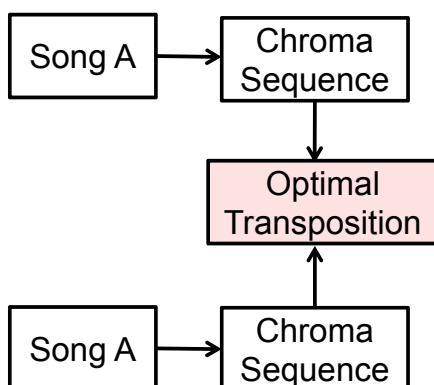


- Feature computation

[Serrà et al., IEEE-TASLP 2009]

Cover Song Identification

How to compare two different songs?

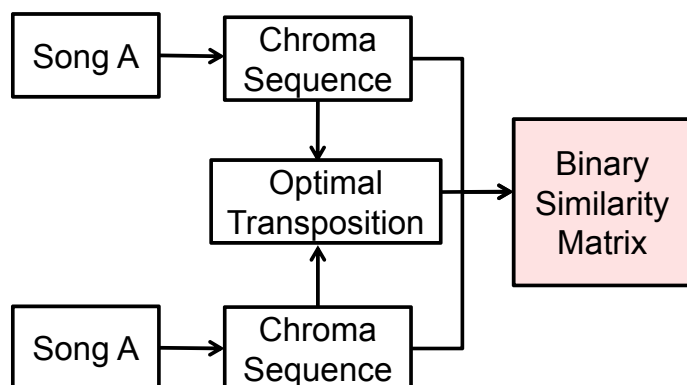


- Feature computation
- Dealing with different keys

[Serrà et al., IEEE-TASLP 2009]

Cover Song Identification

How to compare two different songs?

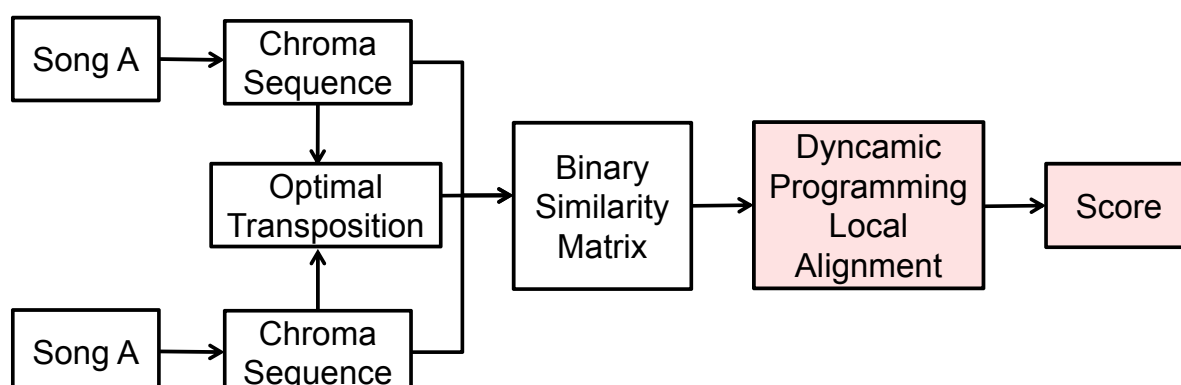


- Feature computation
- Dealing with different keys
- Local similarity measure

[Serrà et al., IEEE-TASLP 2009]

Cover Song Identification

How to compare two different songs?

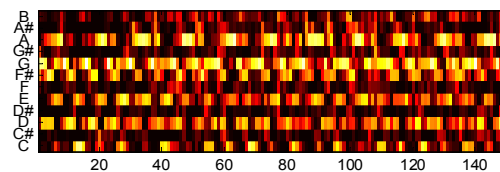


- Feature computation
- Dealing with different keys
- Local similarity measure
- Global similarity measure

[Serrà et al., IEEE-TASLP 2009]

Cover Song Identification

Feature computation



- Chroma features
 - correlates to harmonic progression
 - robust to changes in timbre and instrumentation
 - normalization introduces invariance to dynamics
- Enhancement strategies
 - model for considering harmonics
 - compensation of tuning differences
 - finer resolution (1, 1/2, 1/3 semitone resolution)
 - 12/24/36 dimensional chroma features

[Gómez, PhD 2006]

Cover Song Identification

Dealing with different keys

Bob Dylan – Knockin’ on Heaven’s Door ▶

Avril Lavigne – Knockin’ on Heaven’s Door ▶

- Compute average chroma vectors for each song
- Consider cyclic shifts of the chroma vectors to simulate transpositions
- Determine optimal shift indices so that the shifted chroma vectors are matched with minimal cost
- Transpose the songs accordingly

Cyclic Chroma Shifts

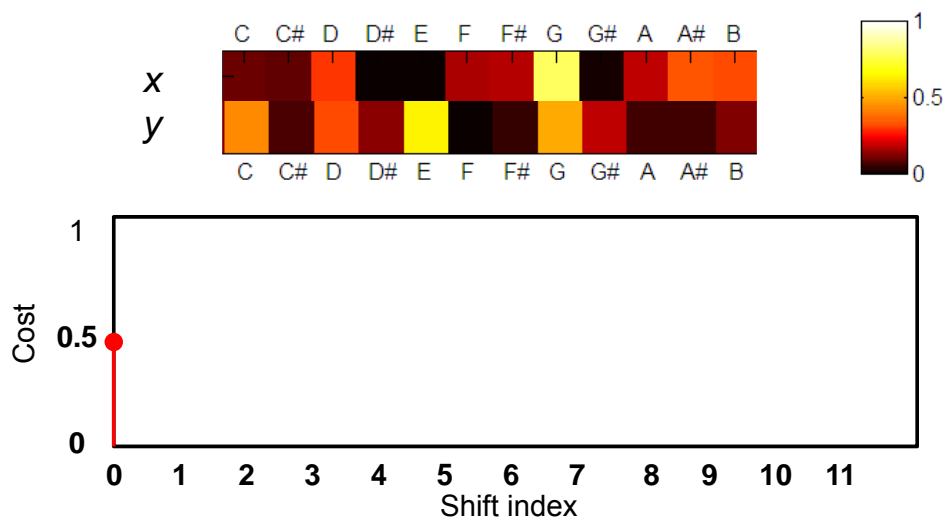
- Feature space: $\mathcal{F} = \mathbb{R}^{12}$
- Chroma vector: $x := (x(1), \dots, x(12))^T \in \mathcal{F}$
- Cyclic shift operator: $\sigma : \mathcal{F} \rightarrow \mathcal{F}$

$$\sigma((x(1), \dots, x(12))^T) := (x(12), x(1), \dots, x(11))^T$$

- Composition of shifts: $\sigma^i(x) = \sigma(\sigma^{i-1}(x)), i \in \mathbb{Z}$
- Note: $\sigma^{12} = \sigma^0$

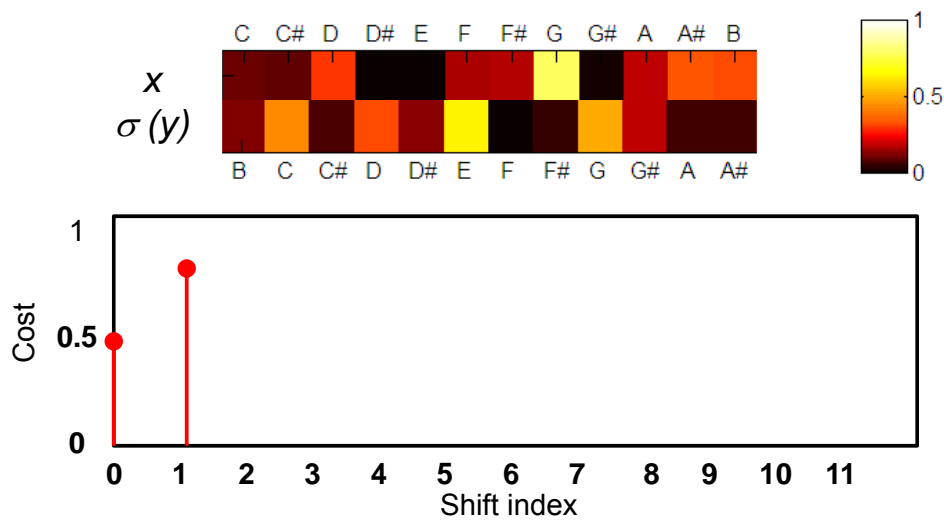
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c : \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



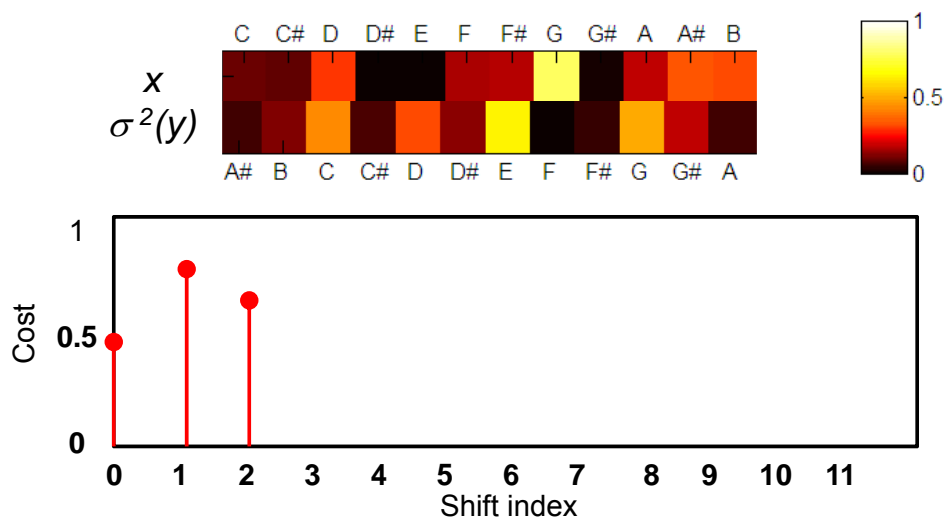
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



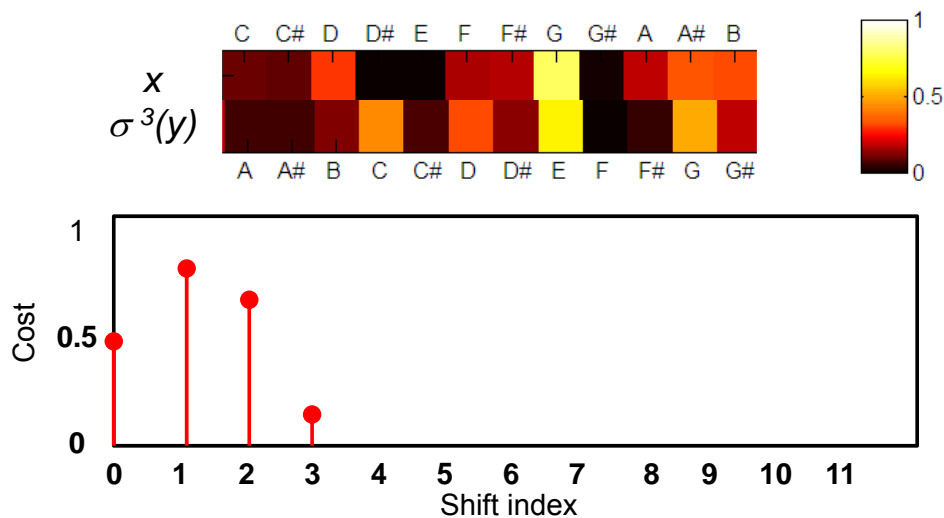
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



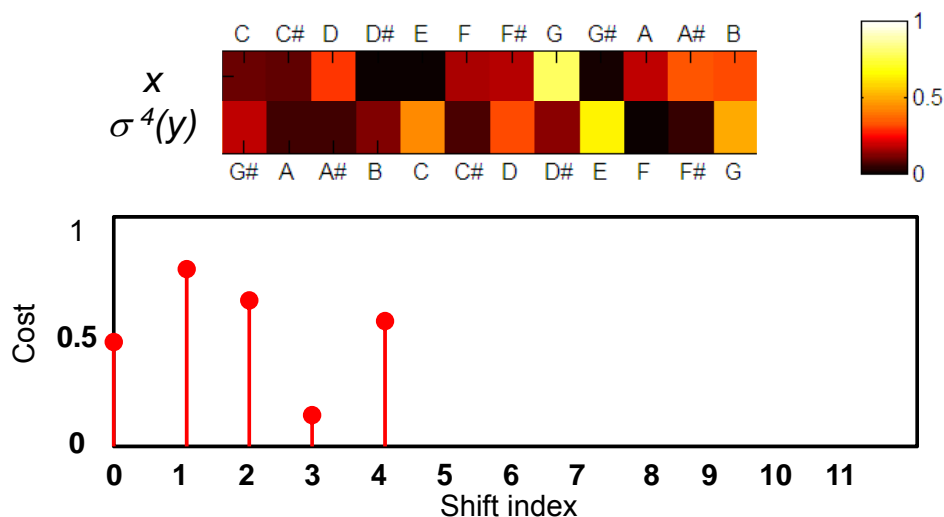
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



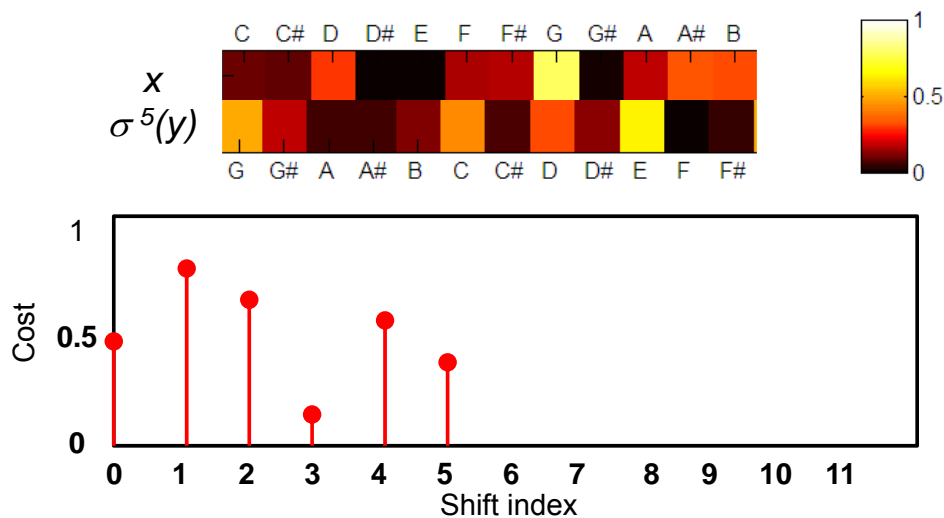
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



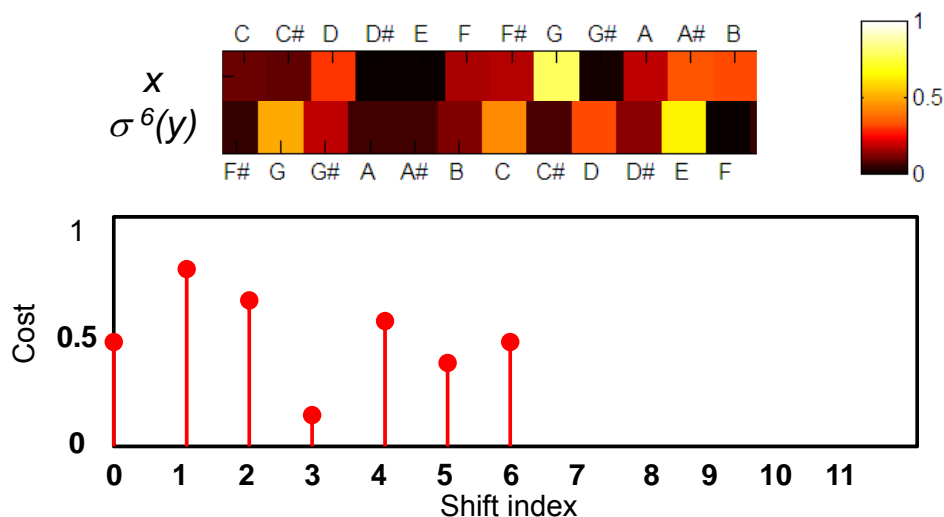
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



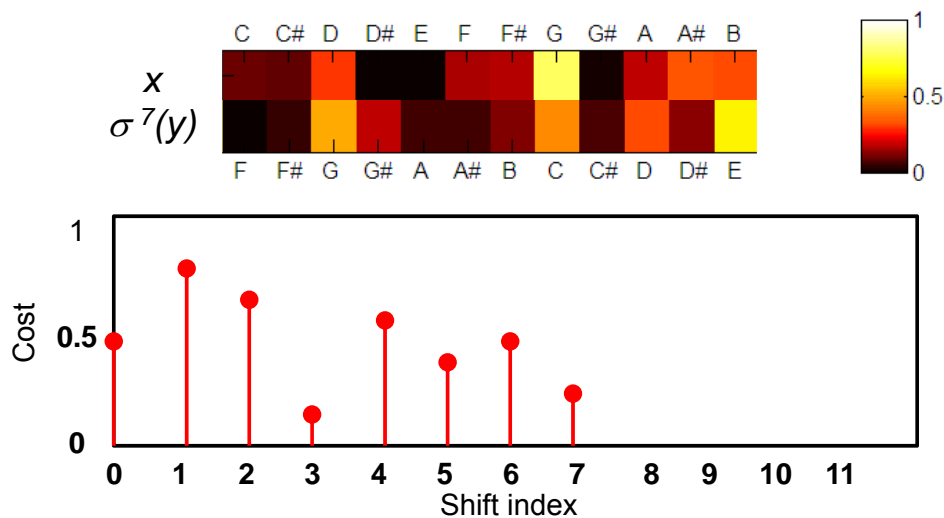
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



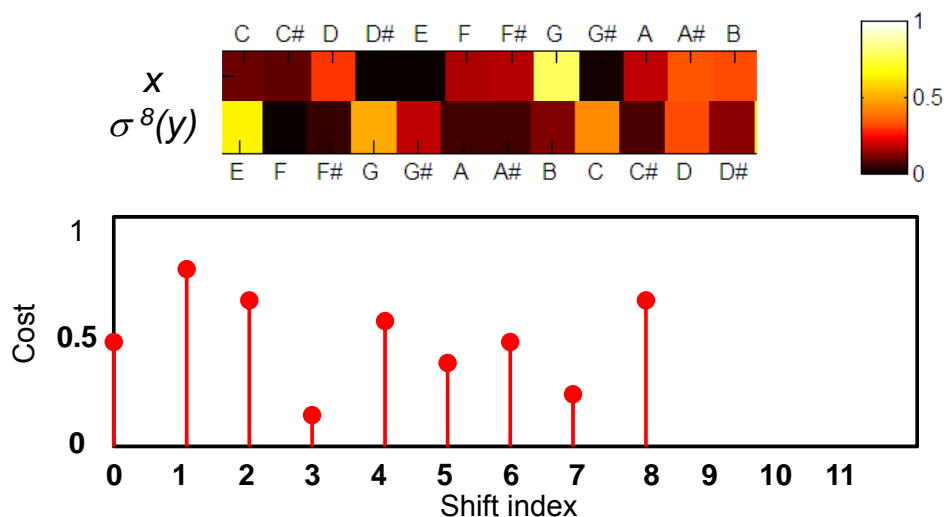
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



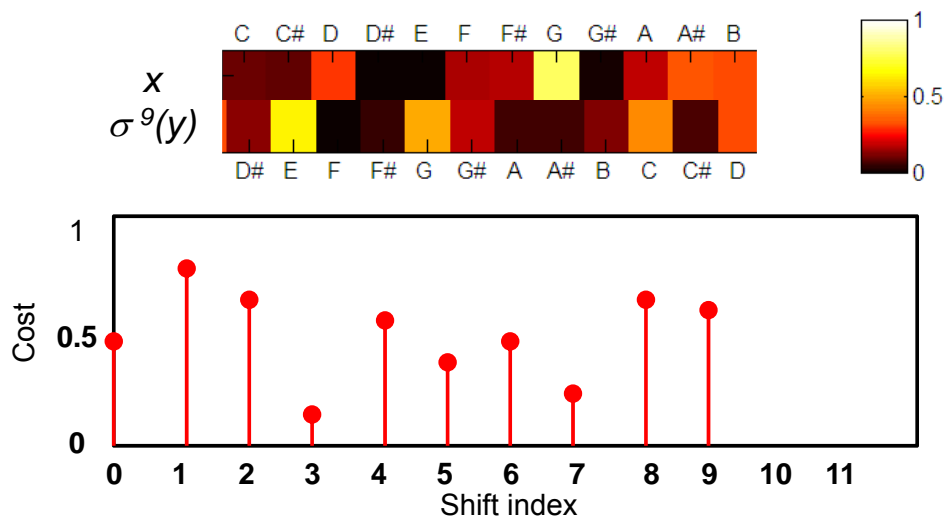
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



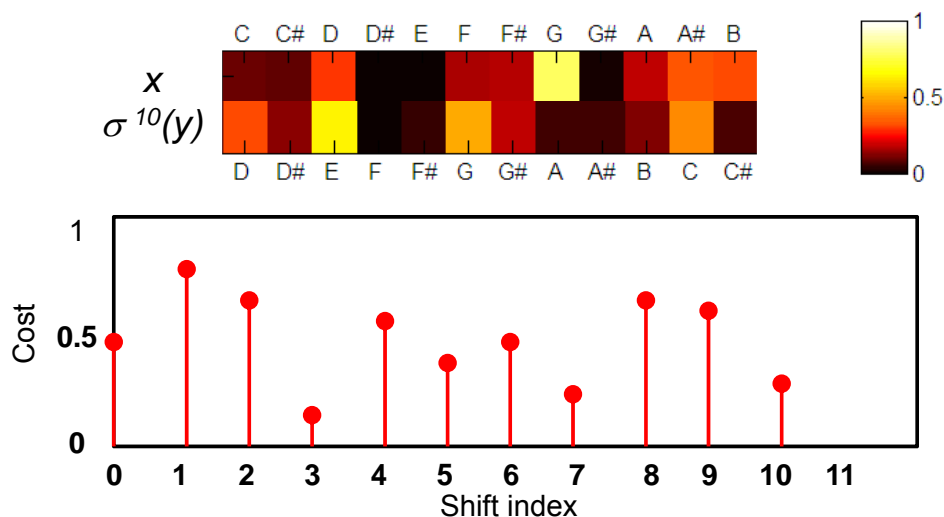
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



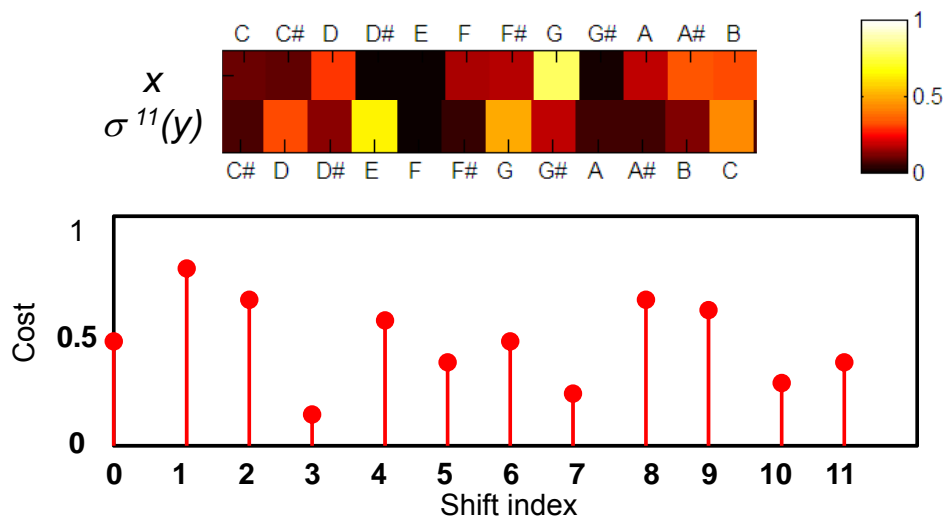
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



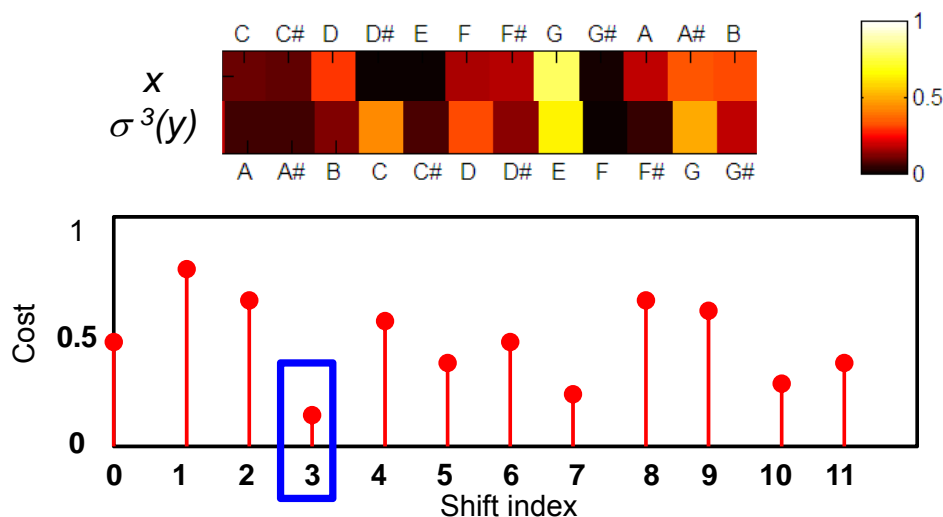
Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y



Cyclic Chroma Shifts

- Given chroma vectors $x, y \in \mathcal{F}$
- Fix a local cost measure $c: \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Compute cost between x and shifted y
- Minimizing shift index: 3



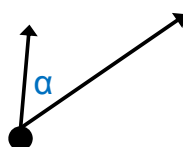
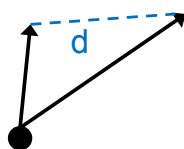
Cyclic Chroma Shifts

- What is a good local cost measure for chroma space?



Cyclic Chroma Shifts

- What is a good local cost measure for chroma space?
Euclidean? Cosine distance?



- Is the chroma space Euclidean?
Probably not!
For example, C is musically closer to G than C#
- Idea: Usage of very coarse **binary cost measure** that indicates the same tonal root

Cyclic Chroma Shifts

- Original local cost measure $c : \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$
- Binary cost measure $c_b : \mathcal{F} \times \mathcal{F} \rightarrow \{0, 1\}$

$$\mu(x, y) := \operatorname{argmin}_{i \in [0:11]} \left(c(x, \sigma^i(y)) \right)$$

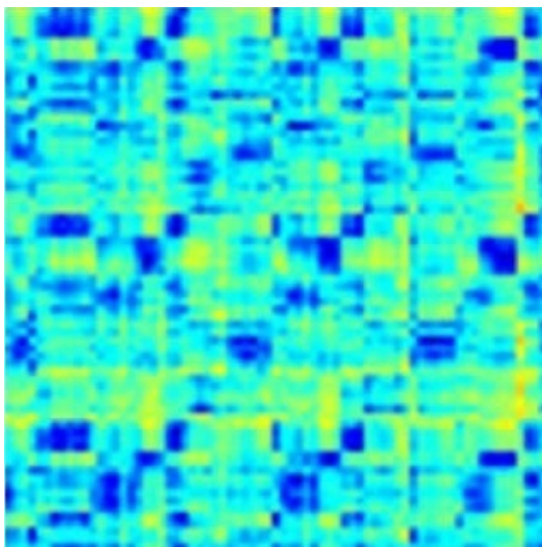
$$c_b(x, y) := \begin{cases} 0 & \text{for } \mu(x, y) = 0 \\ 1 & \text{otherwise} \end{cases}$$

for $x, y \in \mathcal{F}$

[Serrà et al., IEEE-TASLP 2009]

Cyclic Chroma Shifts

Cost matrix based on c



Binary cost matrix based on c_b



Song B

Song B

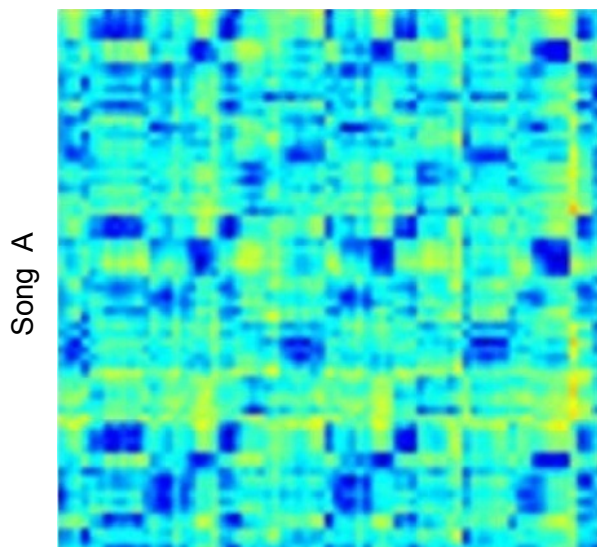
[Serrà et al., IEEE-TASLP 2009]

Cyclic Chroma Shifts

Think positive!

Cost matrix based on c

Binary similarity matrix



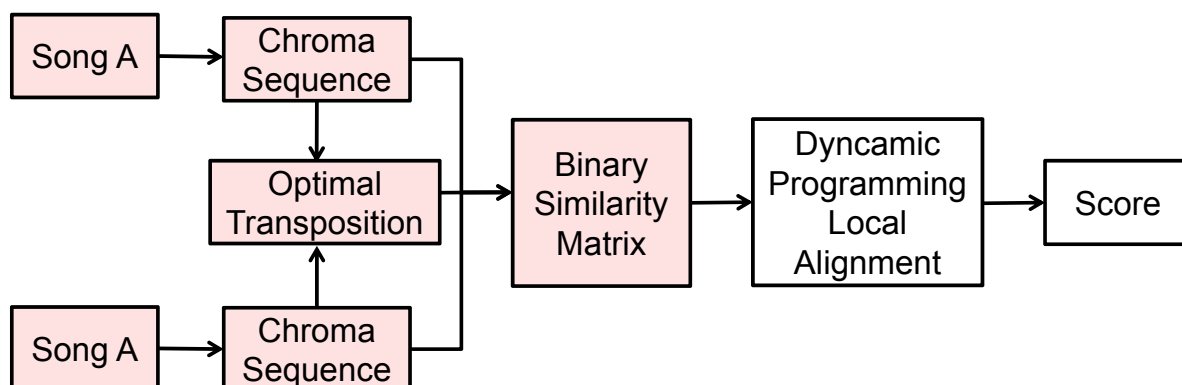
Song B

Song B

[Serrà et al., IEEE-TASLP 2009]

Cover Song Identification

How to compare two different songs?

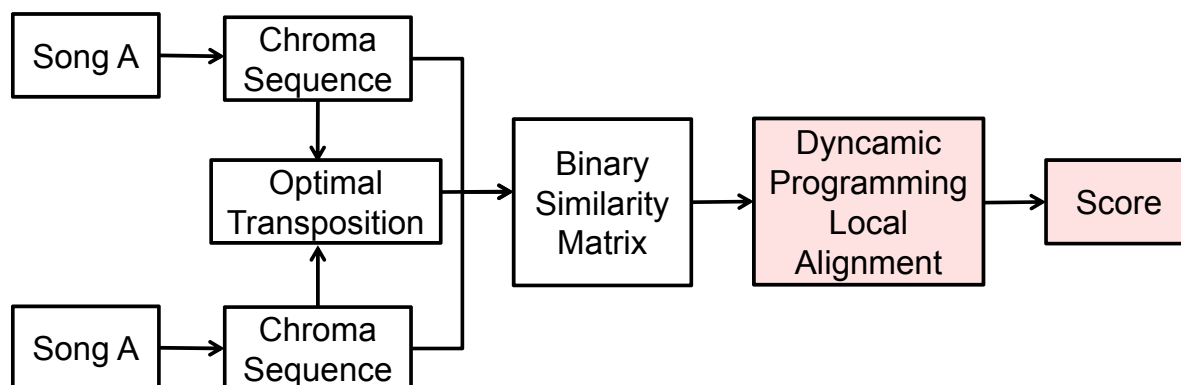


- Feature computation
- Dealing with different keys
- Local similarity measure
- Global similarity measure

[Serrà et al., IEEE-TASLP 2009]

Cover Song Identification

How to compare two different songs?



- Feature computation
- Dealing with different keys
- Local similarity measure
- **Global similarity measure**

[Serrà et al., IEEE-TASLP 2009]

Local Alignment

Assumption:

Two songs are considered as similar if they contain possibly long subsegments that possess a similar harmonic progression

Task:

Let $X=(x_1, \dots, x_N)$ and $Y=(y_1, \dots, y_M)$ be the two chroma sequences of the two given songs, and let S be the resulting similarity matrix. Then find the maximum similarity of a subsequence of X and a subsequence of Y .

Local Alignment

Note:

This problem is also known from bioinformatics.

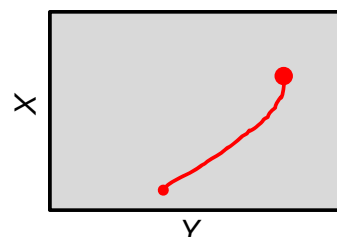
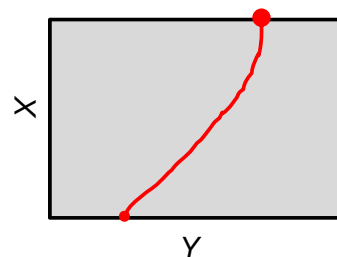
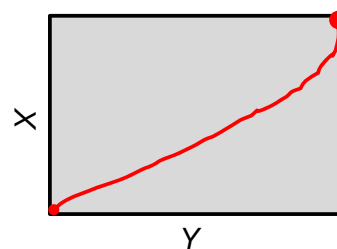
The **Smith-Waterman algorithm** is a well-known algorithm for performing **local sequence alignment**; that is, for determining similar regions between two nucleotide or protein sequences.

Strategy:

We use a variant of the Smith-Waterman algorithm.

Local Alignment

- **Classical DTW**
Global correspondence between X and Y
- **Subsequence DTW**
Subsequence of Y corresponds to X
- **Local Alignment**
Subsequence of Y corresponds to subsequence of X



Local Alignment

Computation of accumulated score matrix D
from given binary similarity (score) matrix S

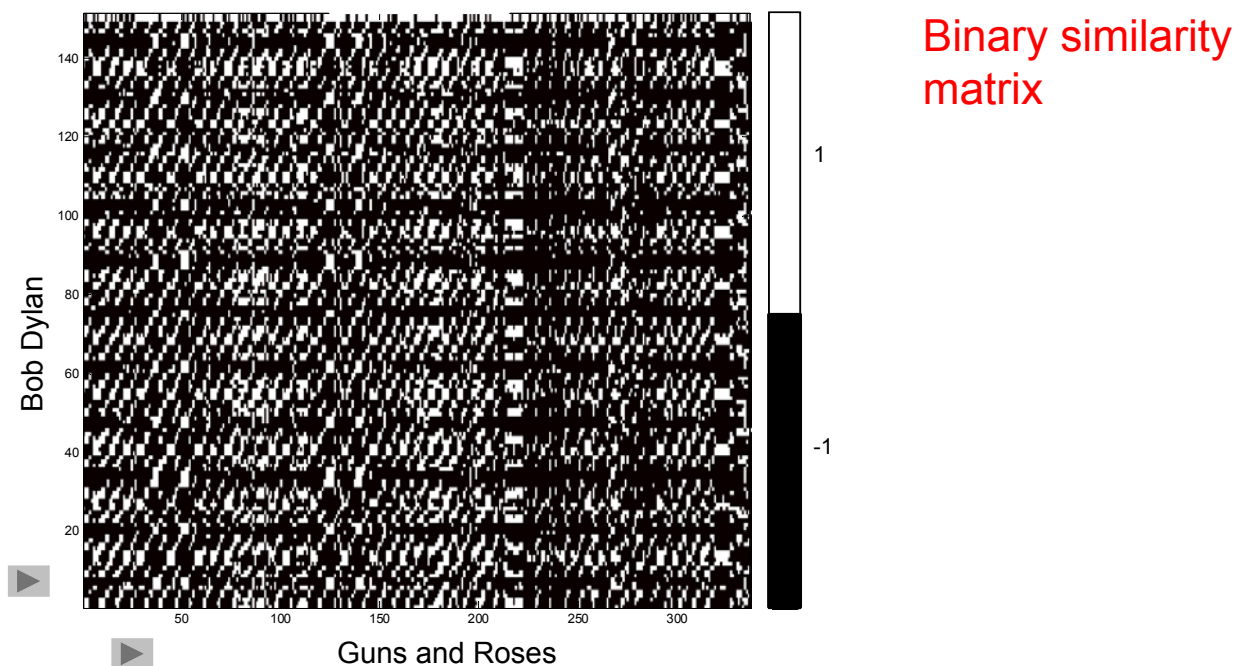
$$D(n, 0) = D(0, m) = 0, \quad n \in [0 : N], m \in [0 : M]$$

$$D(n, m) = \max \begin{cases} 0 \\ D(n-1, m) - g \\ D(n, m-1) - g \\ D(n-1, m-1) + S(n, m) \end{cases}, \quad n, m > 0$$

- Zero-entry allows for jumping to any cell without penalty
- g penalizes “inserts” and “delets” in alignment
- Best local alignment score is the highest value in D
- Best local alignment ends at cell of highest value
- Start is obtained by backtracking to first cell of value zero

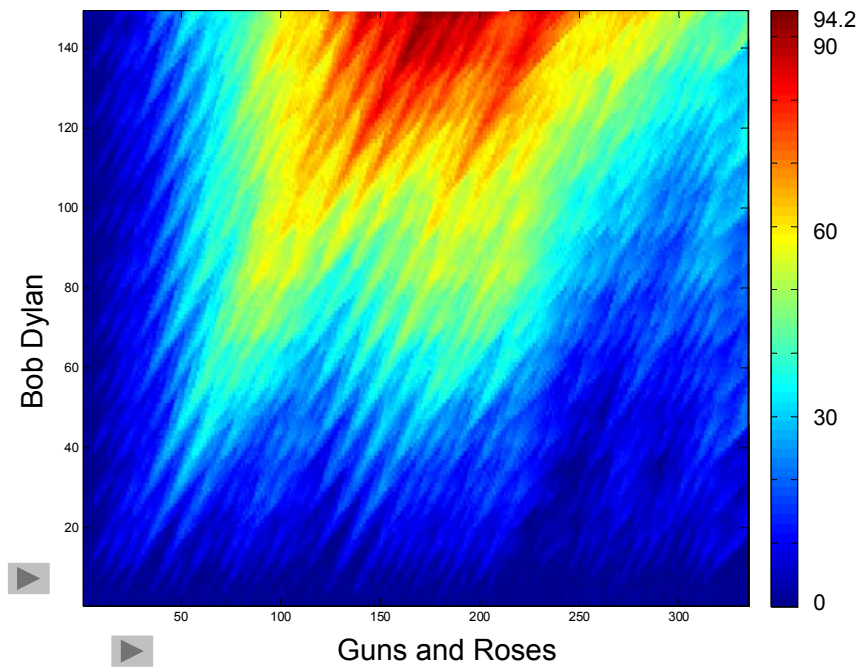
Local Alignment

Example: Knockin' on Heaven's Door



Local Alignment

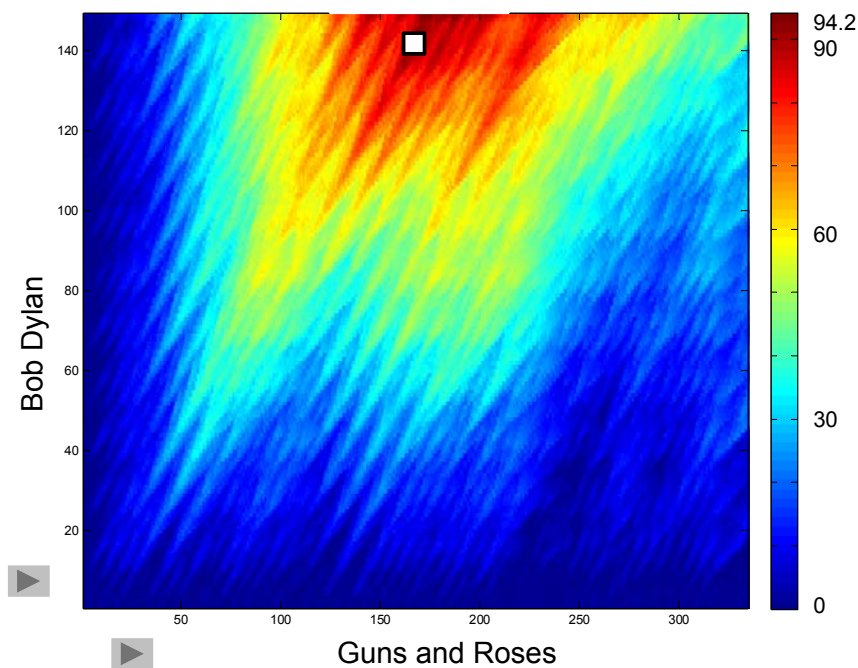
Example: Knockin' on Heaven's Door



Accumulated
score matrix

Local Alignment

Example: Knockin' on Heaven's Door

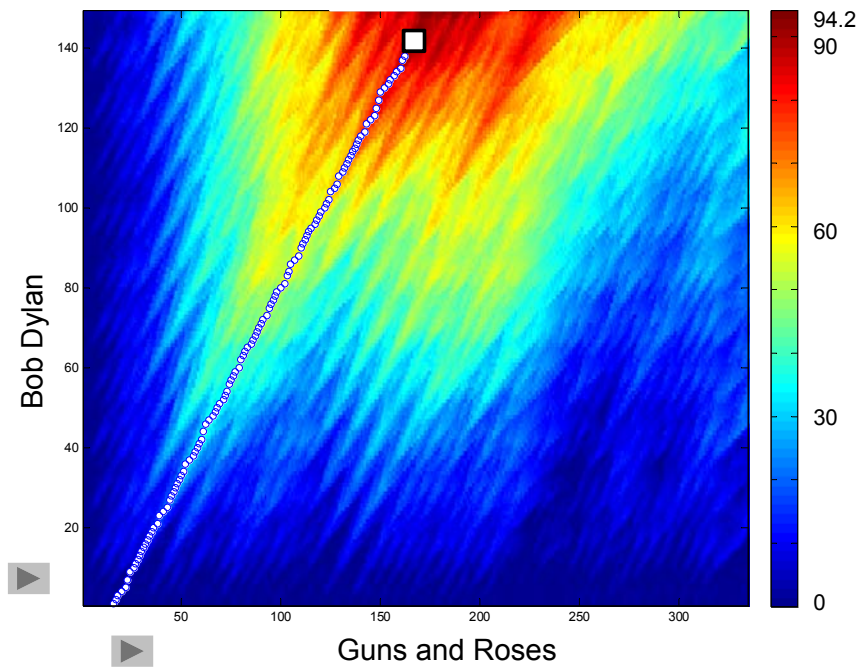


Accumulated
score matrix

Cell with max.
score = 94.2

Local Alignment

Example: Knockin' on Heaven's Door



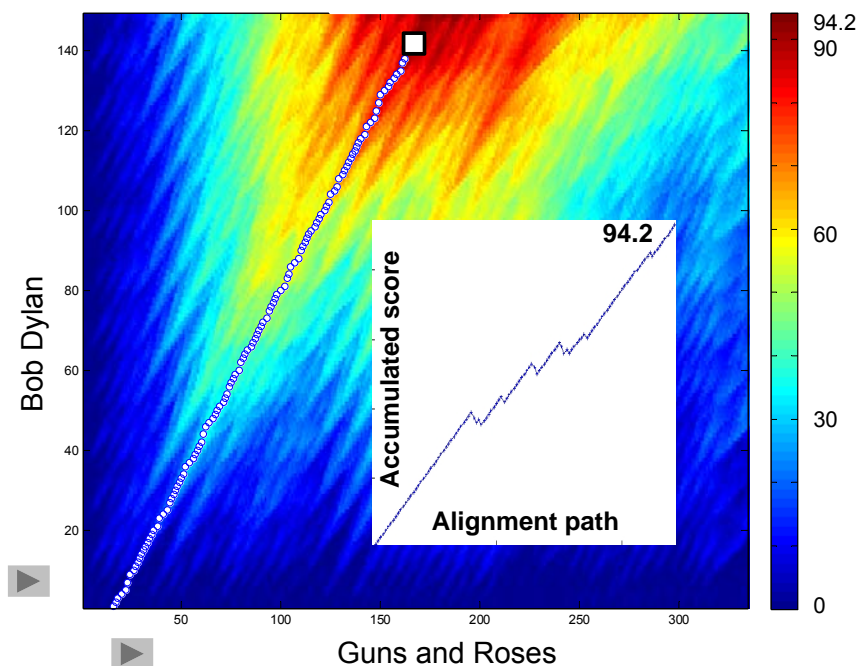
Accumulated score matrix

Cell with max. score = 94.2

Alignment path of maximal score

Local Alignment

Example: Knockin' on Heaven's Door



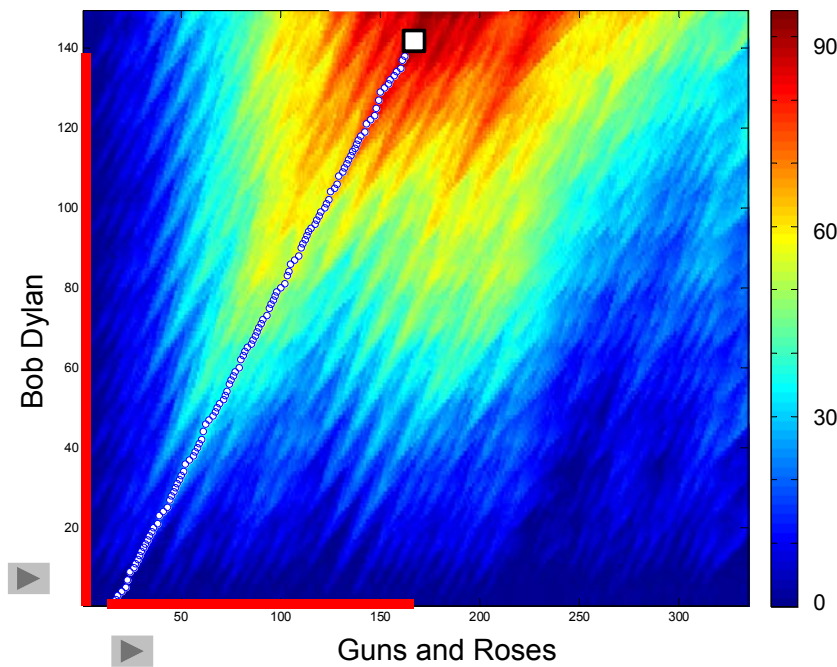
Accumulated score matrix

Cell with max. score = 94.2

Alignment path of maximal score

Local Alignment

Example: Knockin' on Heaven's Door



Accumulated score matrix

Cell with max. score = 94.2

Alignment path of maximal score

Matching subsequences

Cover Song Identification


Query: Bob Dylan – Knockin' on Heaven's Door ▶

Retrieval result:





| Rank | Recording | Score |
|--------|---|-------|
| 1. | Guns and Roses: Knockin' On Heaven's Door | 94.2 |
| 2. | Avril Lavigne: Knockin' On Heaven's Door | 86.6 |
| 3. | Wyclef Jean: Knockin' On Heaven's Door | 83.8 |
| 4. | Bob Dylan: Not For You | 65.4 |
| 5. | Guns and Roses: Patience | 61.8 |
| 6. | Bob Dylan: Like A Rolling Stone | 57.2 |
| 7.-14. | ... | |



Cover Song Identification

Query: AC/DC – Highway To Hell 

Retrieval result:

| Rank | Recording | Score | |
|--------|--|-------|---|
| 1. | AC/DC: Hard As a Rock | 79.2 |  |
| 2. | Hayseed Dixie: Dirty Deeds Done Dirt Cheap | 72.9 |  |
| 3. | AC/DC: Let There Be Rock | 69.6 | |
| 4. | AC/DC: TNT (Live) | 65.0 | |
| 5.-11. | ... | | |
| 12. | Hayseed Dixie: Highway To Hell | 30.4 |  |
| 13. | AC/DC: Highway To Hell Live (live) | 21.0 |  |
| 14. | ... | | |

Conclusions (Cover Song Identification)

- Harmony-based approach
- Binary cost measure a good trade-off between robustness and expressiveness
- Measure is suitable for document retrieval, but seems to be too coarse for audio matching applications
- Every song has to be compared with any other
→ method does not scale to large data collection
- What are suitable indexing methods?

Conclusions (Audio Retrieval)

| Retrieval task | Audio identification | Audio matching | Cover song identification |
|------------------------|-------------------------------|----------------------------|----------------------------------|
| Identification | Concrete audio recording | Different interpretations | Different versions |
| Query | Short fragment (5-10 seconds) | Audio clip (10-40 seconds) | Entire song |
| Retrieval level | Fragment | Fragment | Document |
| Specificity | High | Medium | Medium / Low |
| Features | Spectral peaks (abstract) | Chroma (harmony) | Chroma (harmony) |
| Indexing | Hashing | Inverted lists | No indexing |