CS 583 – Computational Audio

Wayne Snyder
Computer Science Department
Boston University

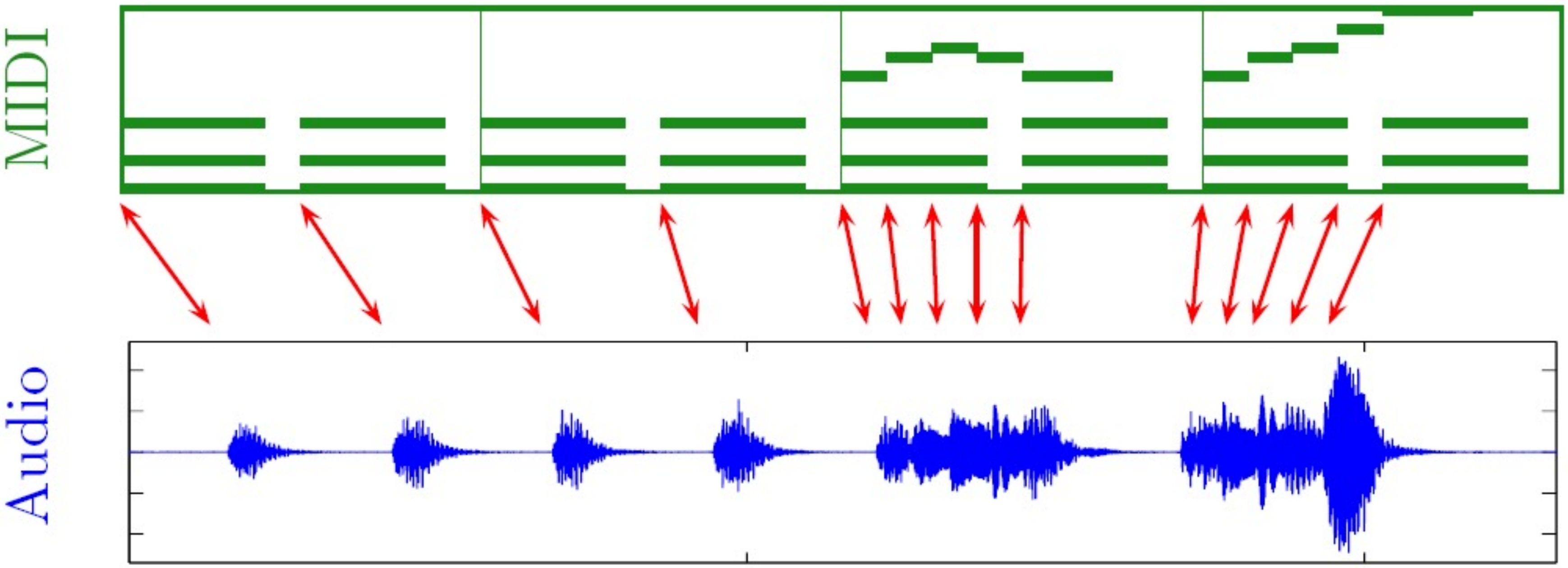
Lecture 17

- Conclusions on Beat Tracking (on notebook)
- Similarity Matrices for Alignment/Synchronization
- Self-similarity Matrices for
  - Structure Analysis
  - Segmentation

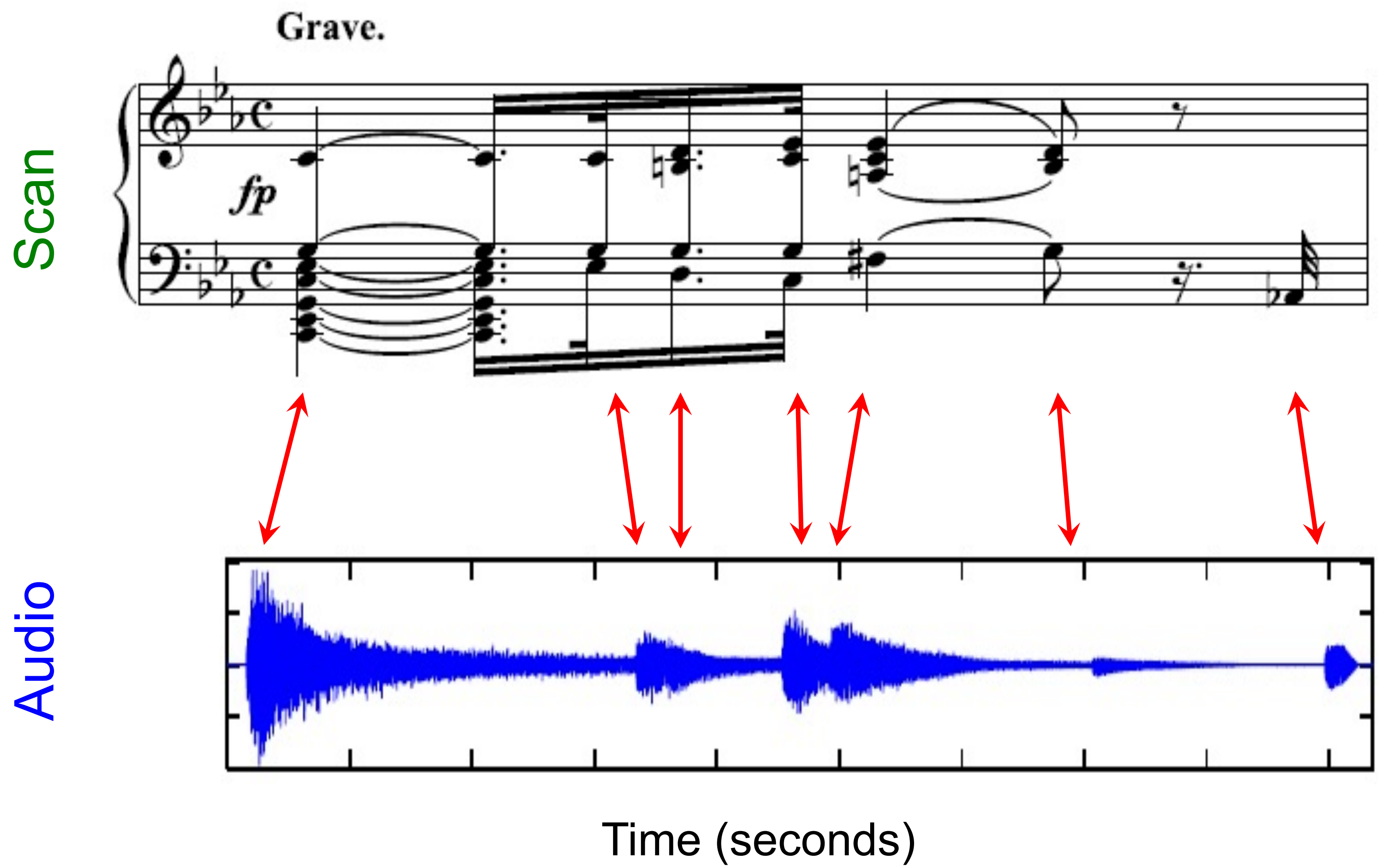
Computer Science

How to compare music with different tempos and timings, and why?

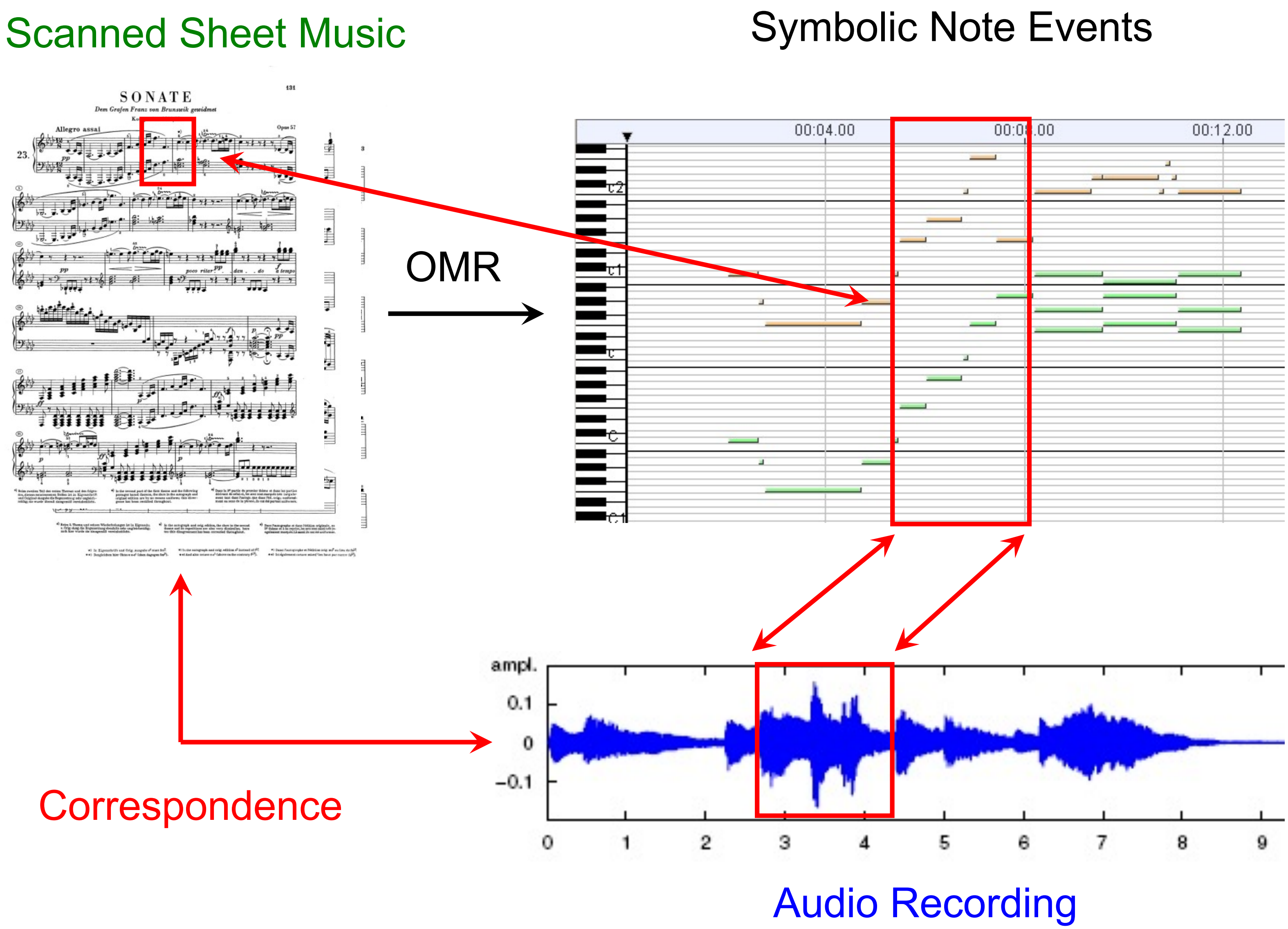
We may want to synchronize two different forms of a signal, e.g., MIDI and WAV File:



Or a score and audio file (e.g., for a Score-following program:

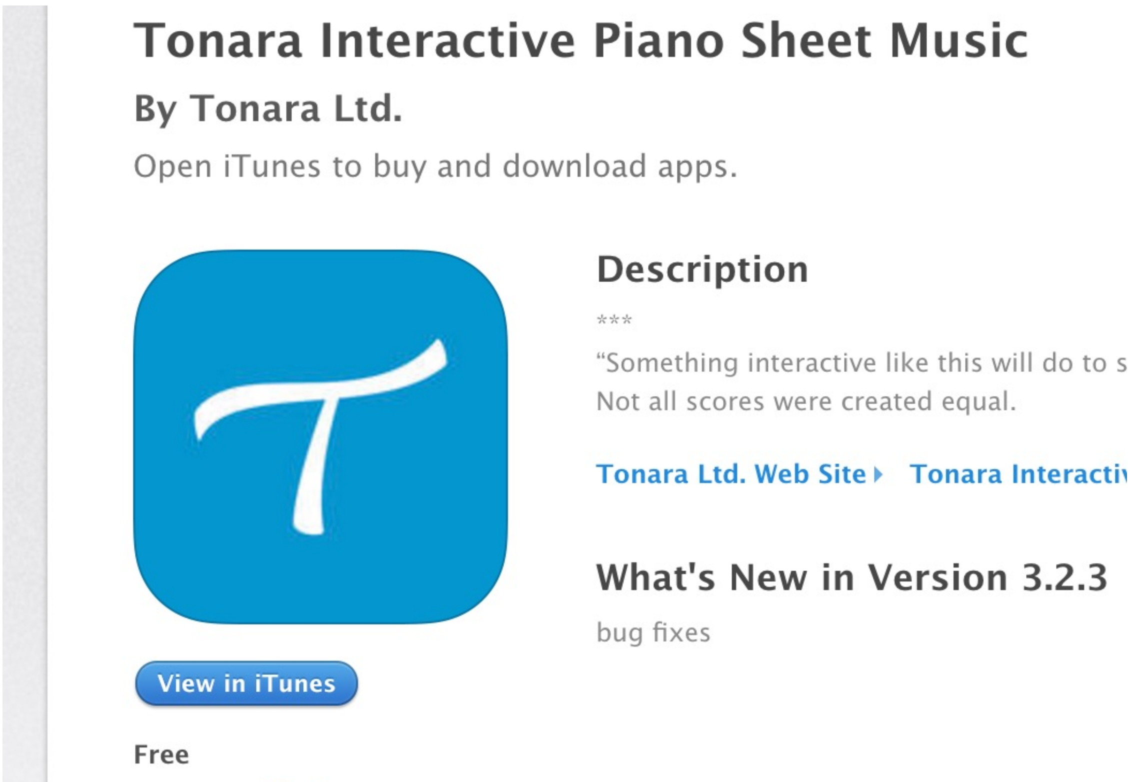


Music Synchronization: Scan-Audio

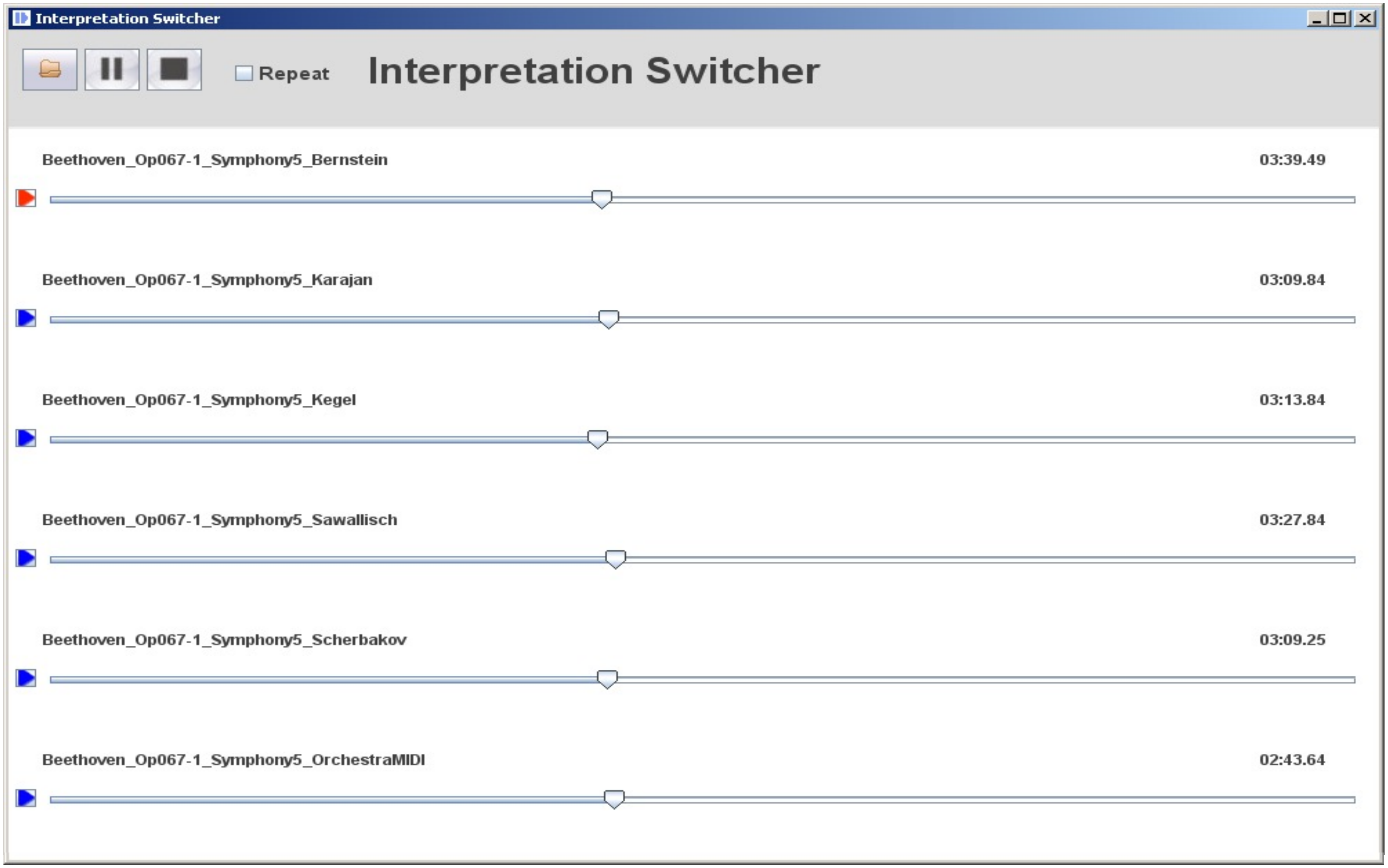


Application: Score Viewer

Tonara iPad App: https://www.youtube.com/user/TonaraSystems



Musicologists and musicians would like to compare multiple versions of the same piece:

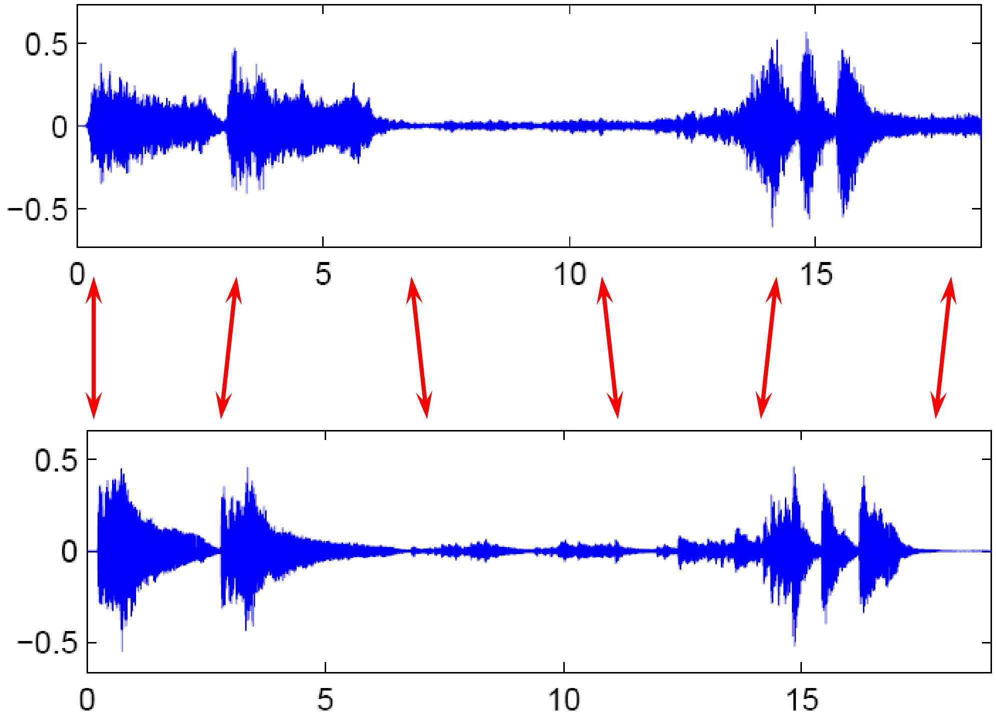


Music Synchronization: Audio-Audio

Beethoven's Fifth

Karajan

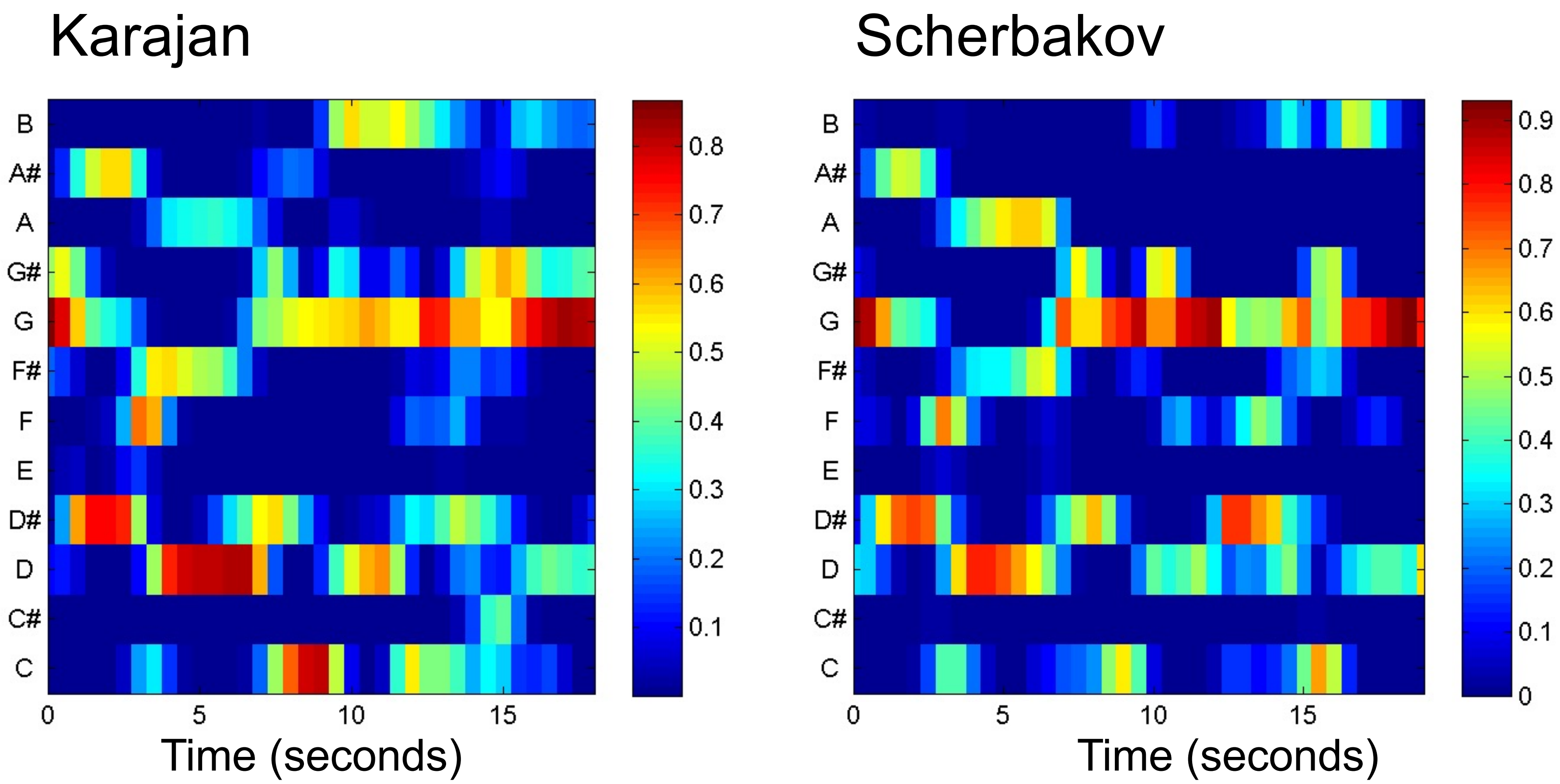
Scherbakov



Synchronization: Karajan → Scherbakov

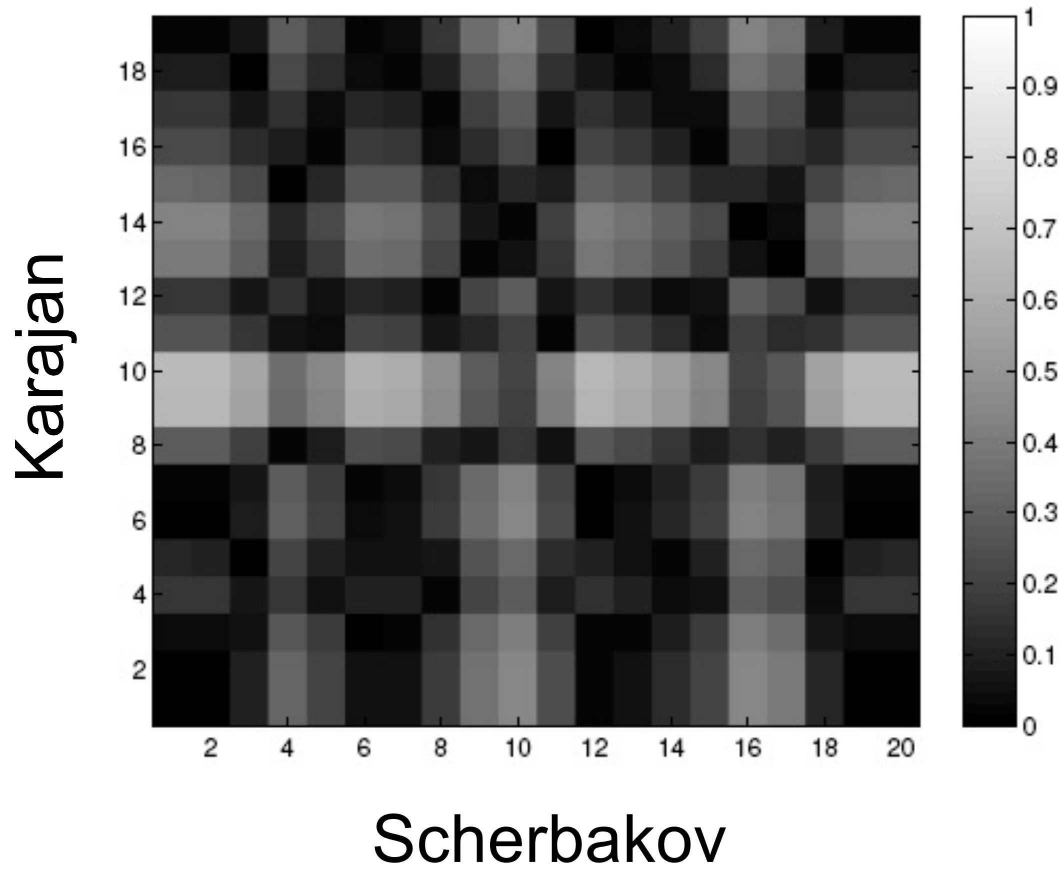
Music Synchronization: Audio-Audio

This is generally done on the chromagram level:



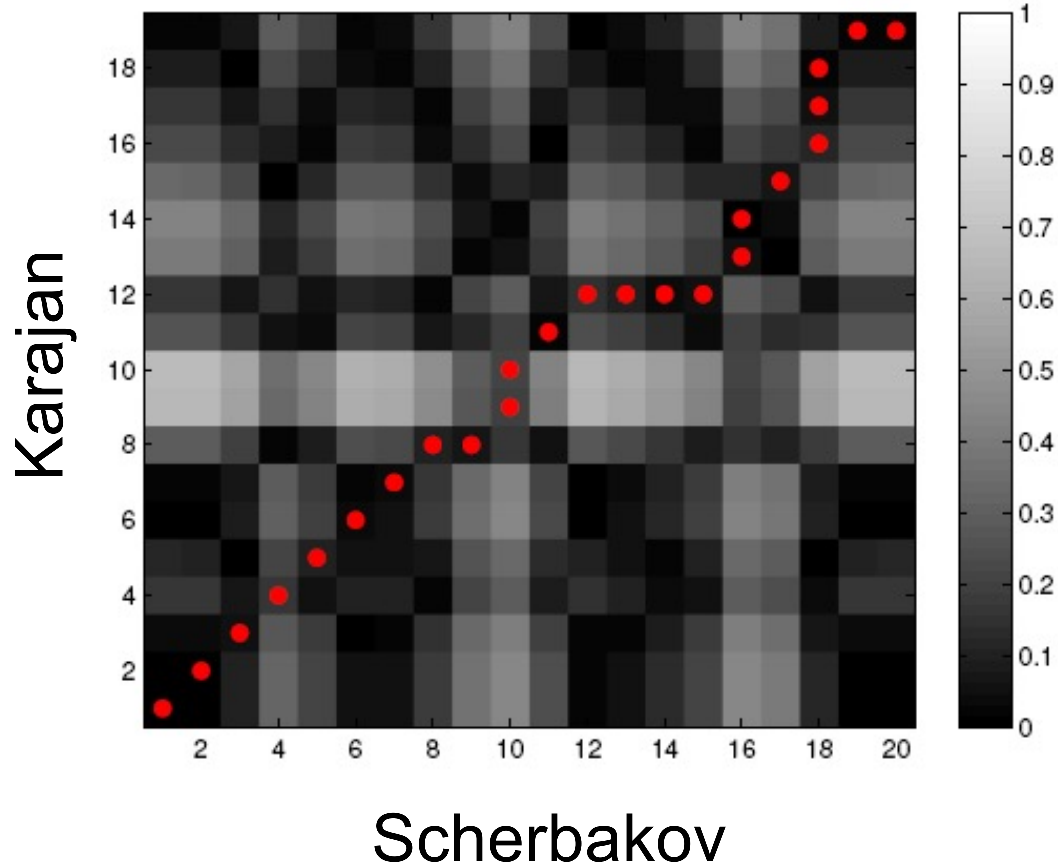
Music Synchronization: Audio-Audio

The standard techniques uses a Similarity/Cost Matrix to compare each chromagraph window in one piece with every window in the other, and measuring their distance:



Music Synchronization: Audio-Audio

The result of this analysis is a cost-minimizing warping path which gives the alignment:



Music Synchronization: Audio-Audio

Let's consider a simple related problem as a warmup to the issues….

Approximate String Matching Problem:

Given two strings $a_1a_2….a_n$ and $b_1….b_m$ what is the minimum edit distance between the two strings relative to a set of edit operations with costs, e.g.,

delete a character (from either string)	cost = 1
change a character	cost = 1

The goal is to minimize the total cost to convert one string to another.

Example: SNEIDER SNYDER

SNEIDER $\xrightarrow[\text{delete}]{}$ SNIDER $\xrightarrow[\text{change}]{}$ SNYDER   Total cost = 2

Example: SCHNIDEIR SNYDER

SCHNIDEIR $\xrightarrow[\text{delete}]{}$ SHNIDEIR $\xrightarrow[\text{delete}]{}$ SNIDEIR $\xrightarrow[\text{change}]{}$ SNYDEIR $\xrightarrow[\text{delete}]{}$ SNYDER

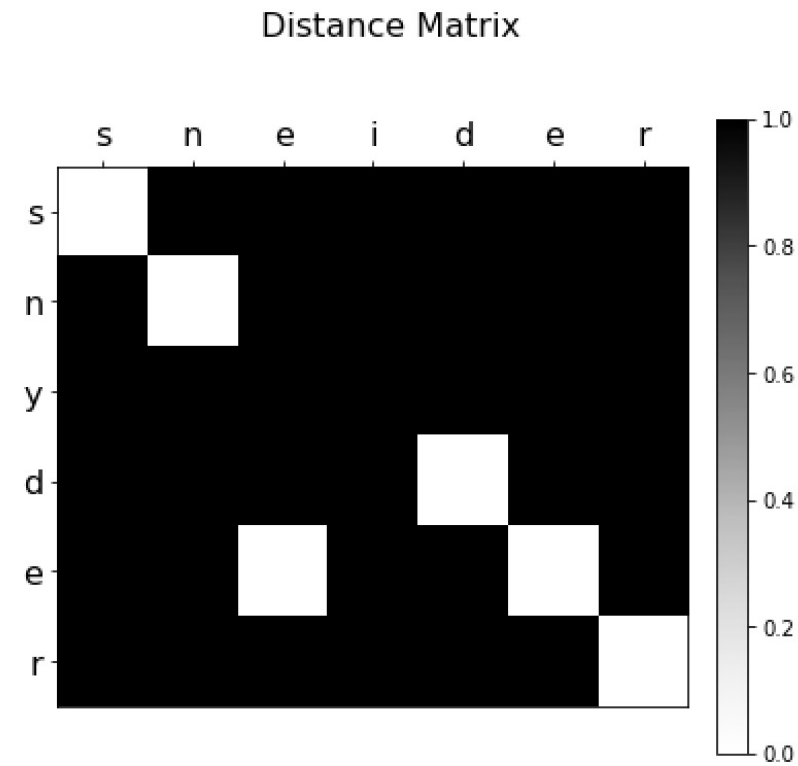
total cost = 4

Music Synchronization: Audio-Audio

How to compute minimum cost path between $a_1a_2....a_n$ and $b_1....b_m$?

Create n x m Distance Matrix, giving distance between each pair of letters; supposing cost of a change = 1 for all all pairs

```python
s = "snyder"
t = "sneider"

Rows = list(s)      # explode string to list
Cols = list(t)

# calculate distance

def dist(a,b):
    if(a==b):
        return 0
    else:
        return 1

# Create Distance Matrix

D = [[0 for x in range(len(Cols))]
        for y in range(len(Rows))]

for k in range(len(Rows)):
    for m in range(len(Cols)):
        D[k][m] = dist(Rows[k],Cols[m])
```




```
0.0   1.0   1.0   1.0   1.0   1.0   1.0
1.0   0.0   1.0   1.0   1.0   1.0   1.0
1.0   1.0   1.0   1.0   1.0   1.0   1.0
1.0   1.0   1.0   1.0   0.0   1.0   1.0
1.0   1.0   0.0   1.0   1.0   0.0   1.0
1.0   1.0   1.0   1.0   1.0   1.0   0.0
```

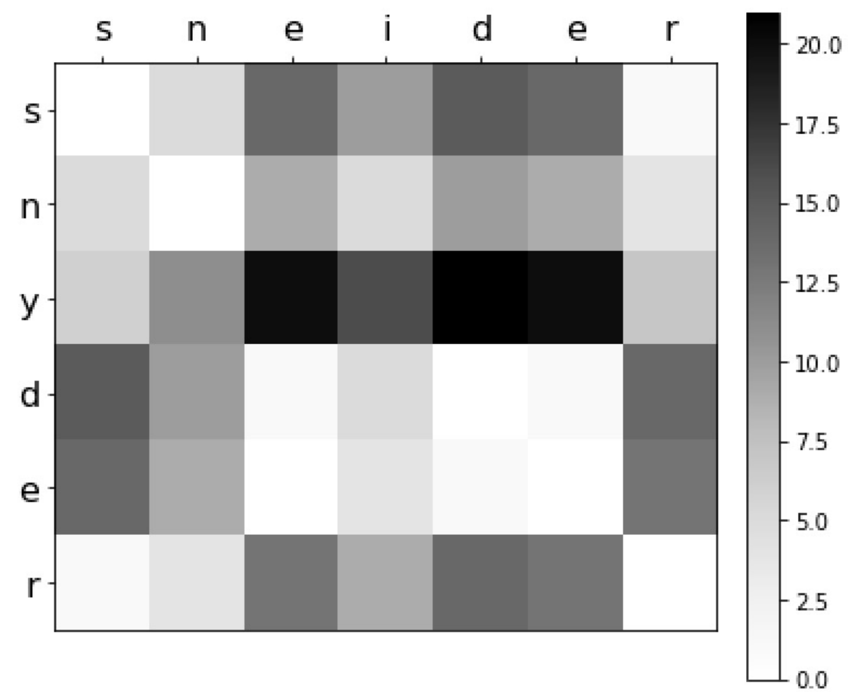
Music Synchronization: Audio-Audio

How to compute minimum cost path between $a_1a_2....a_n$ and $b_1....b_m$?

Create n x m Distance Matrix, giving distance between each pair of letters;
supposing cost of a change = distance between letters

```
s = "snyder"
t = "sneider"

Rows = list(s)      # explode string to list
Cols = list(t)

# calculate distance

def dist(a,b):
    return abs(ord(a)-ord(b))

# Create Distance Matrix

D = [[0 for x in range(len(Cols))]
        for y in range(len(Rows))]

for k in range(len(Rows)):
    for m in range(len(Cols)):
        D[k][m] = dist(Rows[k],Cols[m])
```

Distance Matrix

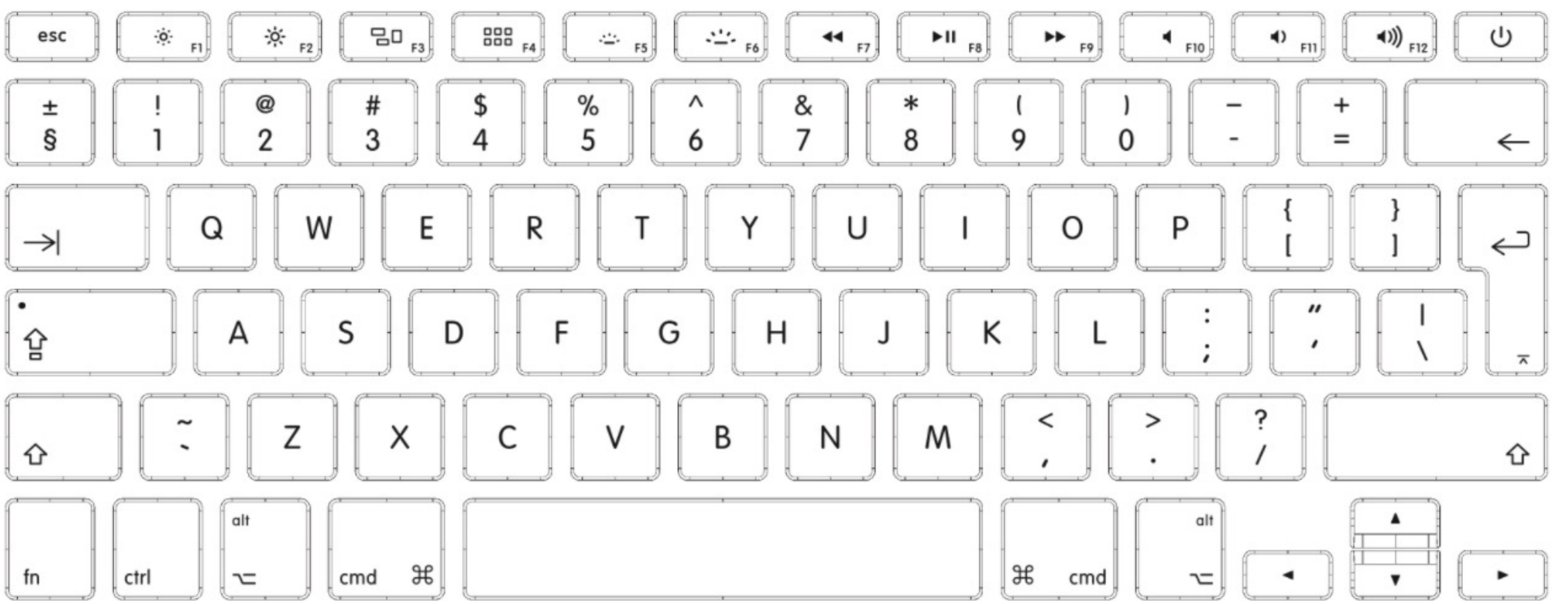


0.0	5.0	14.0	10.0	15.0	14.0	1.0
5.0	0.0	9.0	5.0	10.0	9.0	4.0
6.0	11.0	20.0	16.0	21.0	20.0	7.0
15.0	10.0	1.0	5.0	0.0	1.0	14.0
14.0	9.0	0.0	4.0	1.0	0.0	13.0
1.0	4.0	13.0	9.0	14.0	13.0	0.0

Music Synchronization: Audio-Audio

How to compute minimum cost path between $a_1 a_2 \ldots a_n$ and $b_1 \ldots b_m$?

Create n x m Distance Matrix, giving distance between each pair of letters;
supposing cost of a change = distance between letters on a keyboard

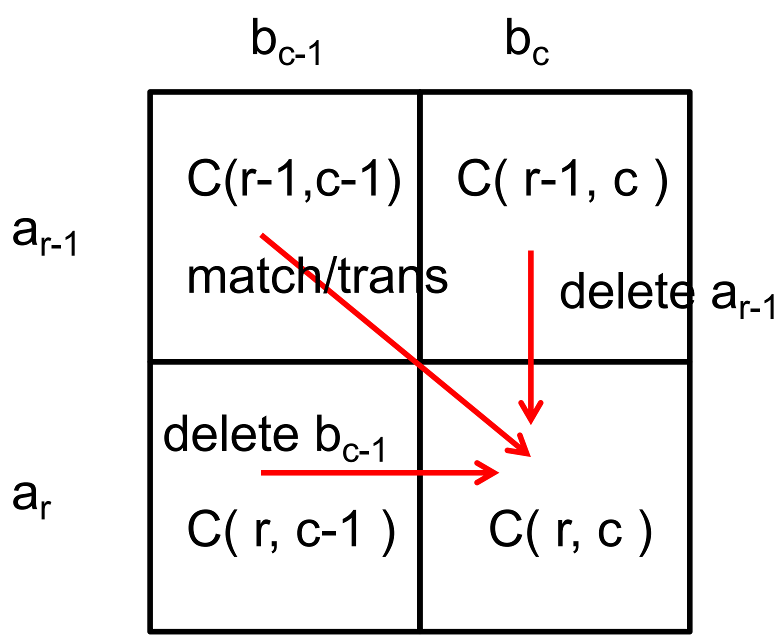


Music Synchronization: Audio-Audio

How to compute minimum cost path between $a_1a_2\ldots a_n$ and $b_1\ldots b_m$?

Next, create a Cost Matrix, giving the minimum cost to arrive at a particular cell in the matrix; for cell (r,c), this is the minimum cost of matching
$s = a_1a_2\ldots a_r$ and $t = b_1\ldots b_c$

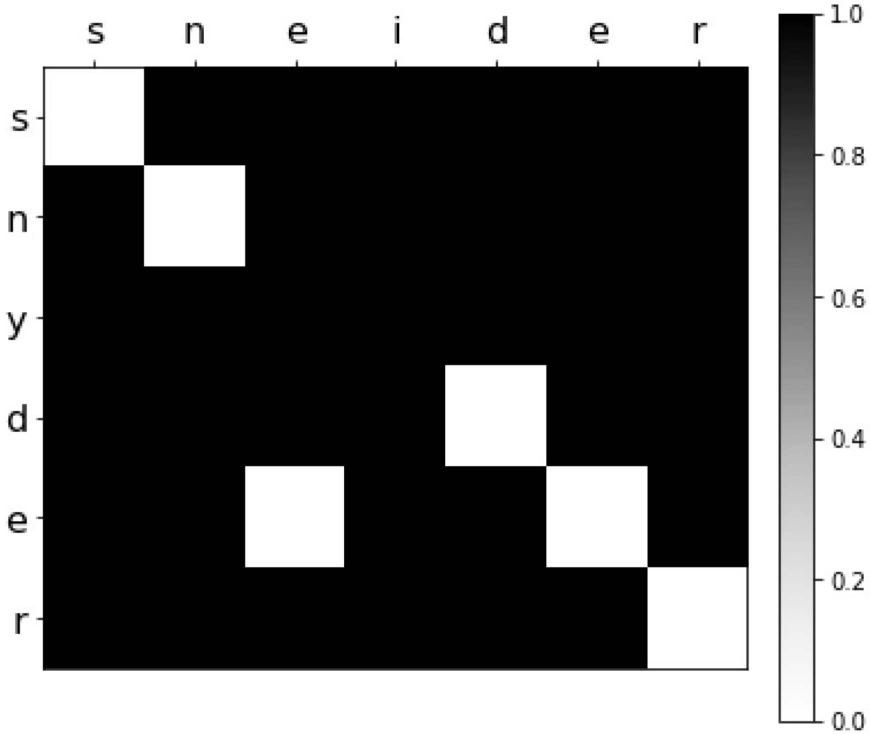
```
C = [[0 for x in range(len(Cols))] for y in range(len(Rows))]

C[0][0] = D[0][0]

for c in range(1,len(Cols)):
    C[0][c] = C[0][c-1] + 1

for r in range(1,len(Rows)):
    C[r][0] = C[r-1][0] + 1

for r in range(1,len(Rows)):
    for c in range(1,len(Cols)):
        left    = C[r][c-1] + 1
        up      = C[r-1][c] + 1
        upleft  = C[r-1][c-1] + D[r-1][c-1]
        C[k][m] = min(left,up,upleft)
```



Music Synchronization: Audio-Audio

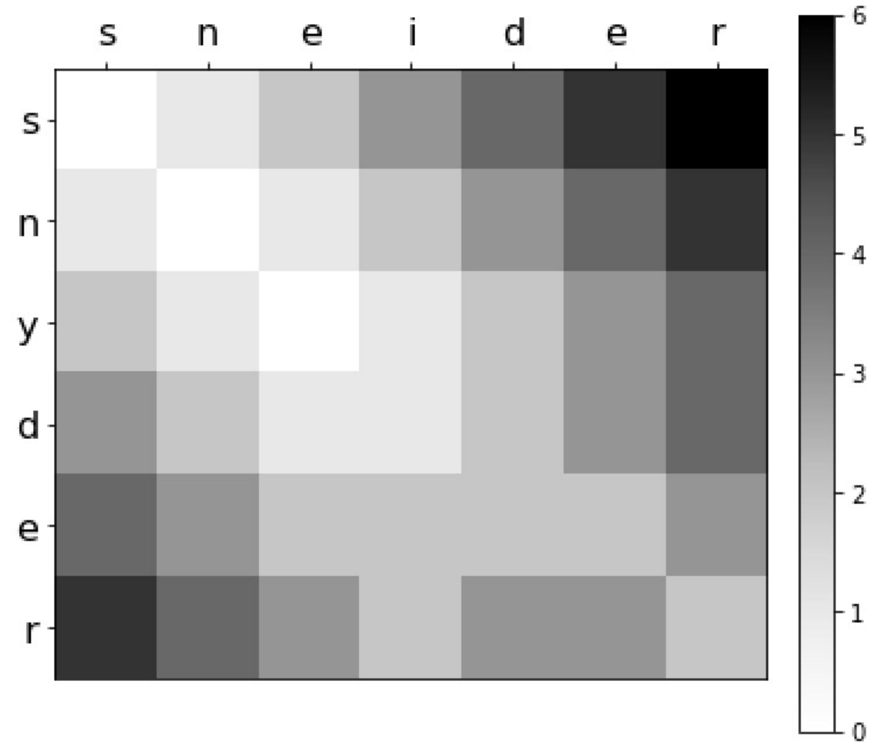
Next, create a Cost Matrix…….

Distance Matrix



0.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	0.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	0.0	1.0	1.0
1.0	1.0	0.0	1.0	1.0	0.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	0.0

Cost Matrix



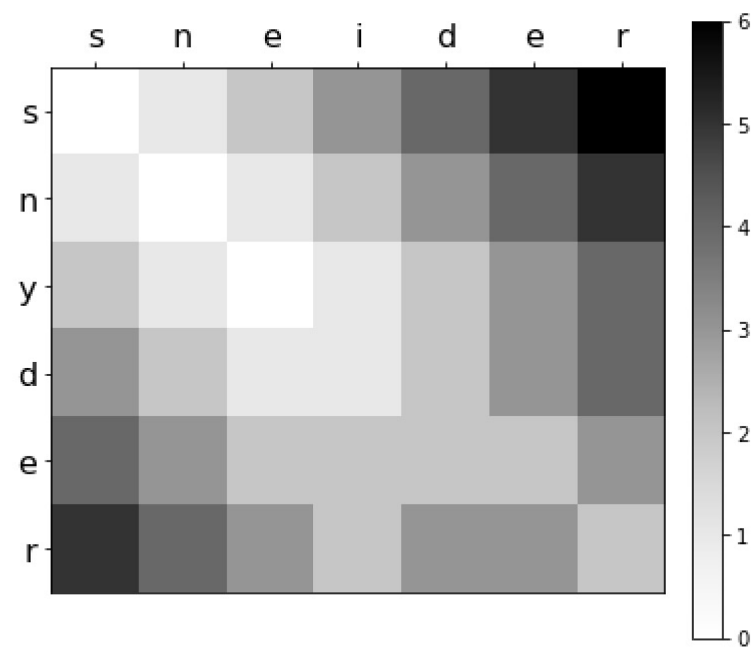
0.0	1.0	2.0	3.0	4.0	5.0	6.0
1.0	0.0	1.0	2.0	3.0	4.0	5.0
2.0	1.0	0.0	1.0	2.0	3.0	4.0
3.0	2.0	1.0	1.0	2.0	3.0	4.0
4.0	3.0	2.0	2.0	2.0	2.0	3.0
5.0	4.0	3.0	2.0	3.0	3.0	2.0

Music Synchronization: Audio-Audio

Finally, while creating the Cost Matrix, keep track of the minimum path from the upper left to the lower right corner:

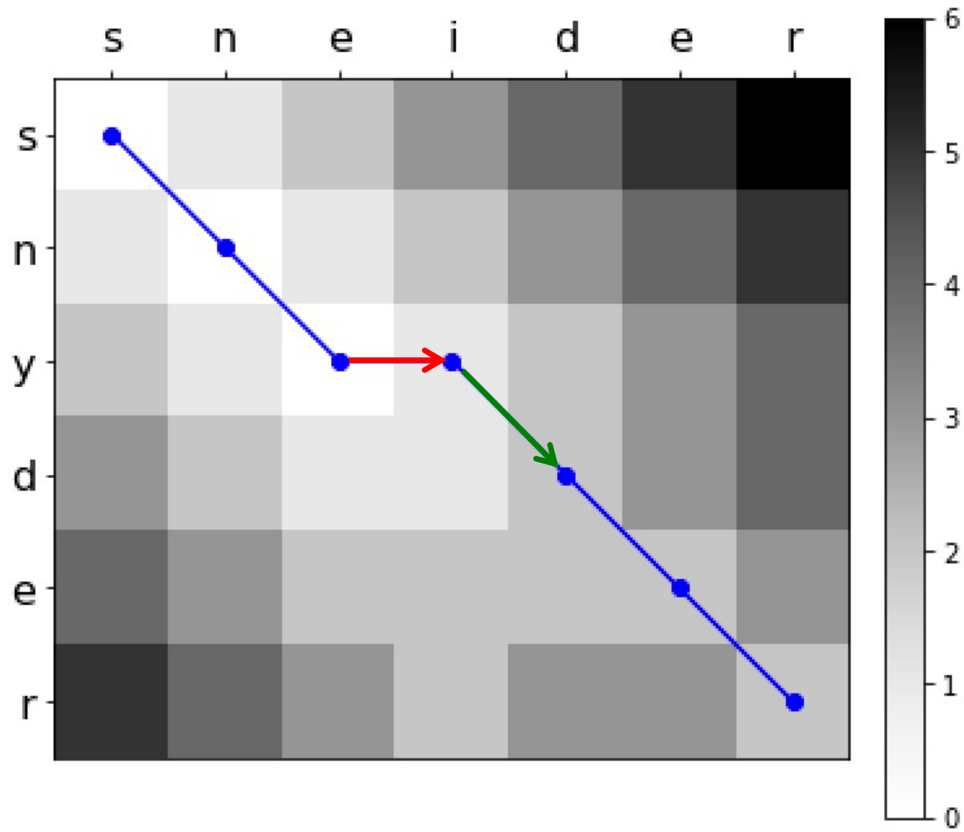
SNEIDER $\xrightarrow[\text{delete}]{}$ **SNIDER** $\xrightarrow[\text{change}]{}$ **SNYDER** **Total cost = 2**

Cost Matrix



0.0	1.0	2.0	3.0	4.0	5.0	6.0
1.0	0.0	1.0	2.0	3.0	4.0	5.0
2.0	1.0	0.0	1.0	2.0	3.0	4.0
3.0	2.0	1.0	1.0	2.0	3.0	4.0
4.0	3.0	2.0	2.0	2.0	2.0	3.0
5.0	4.0	3.0	2.0	3.0	3.0	2.0

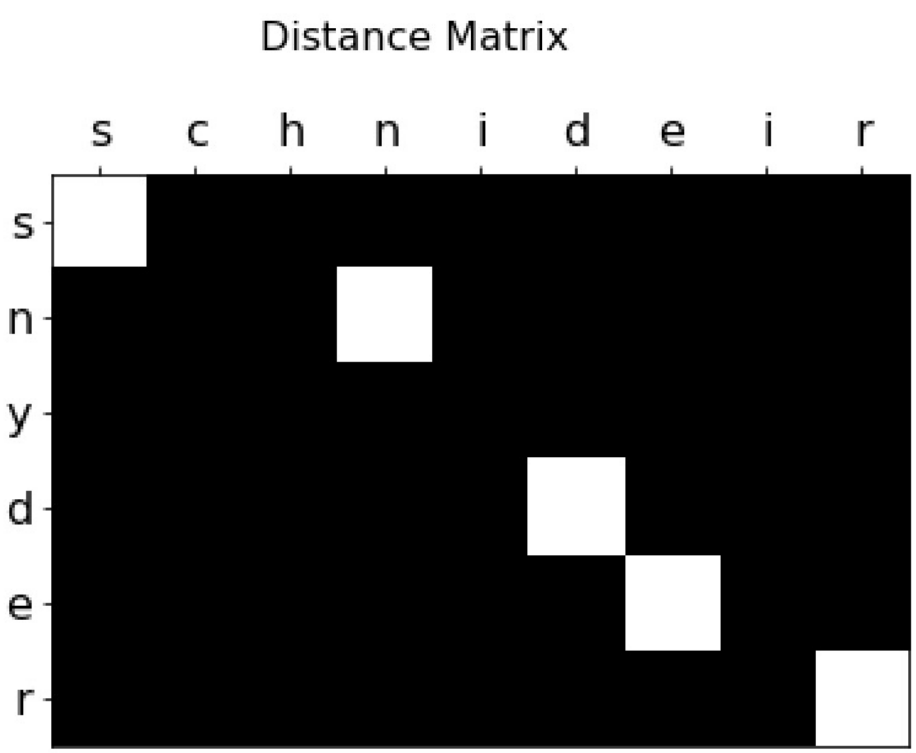
Cost Matrix with Least-Cost Path (cost=2)

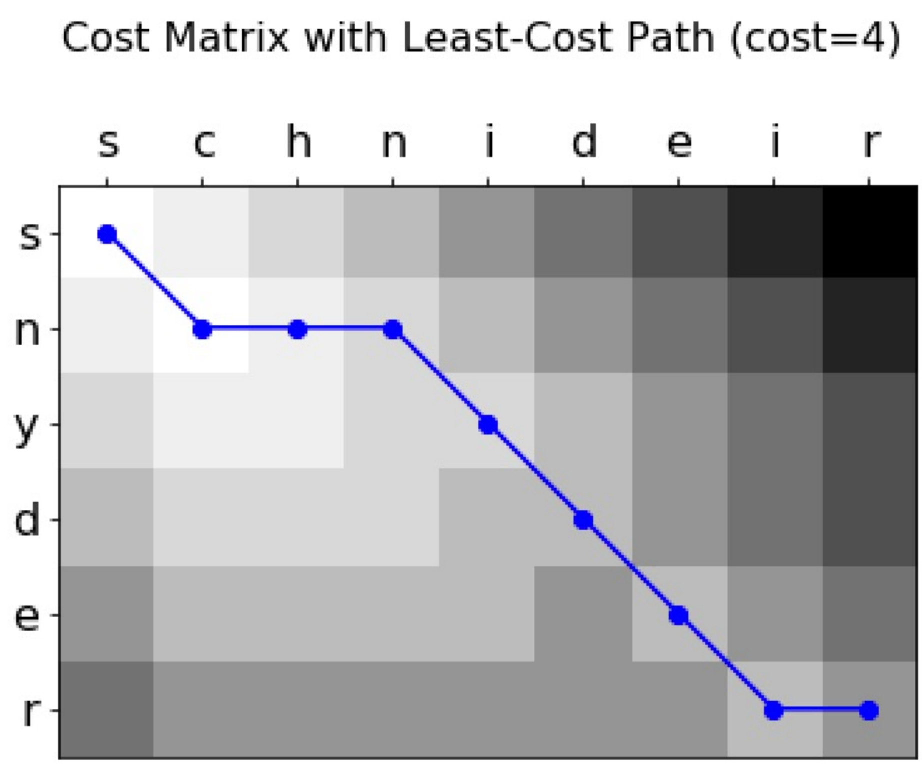


Music Synchronization: Audio-Audio

Finally, while creating the Cost Matrix, keep track of the minimum path from the upper left to the lower right corner:

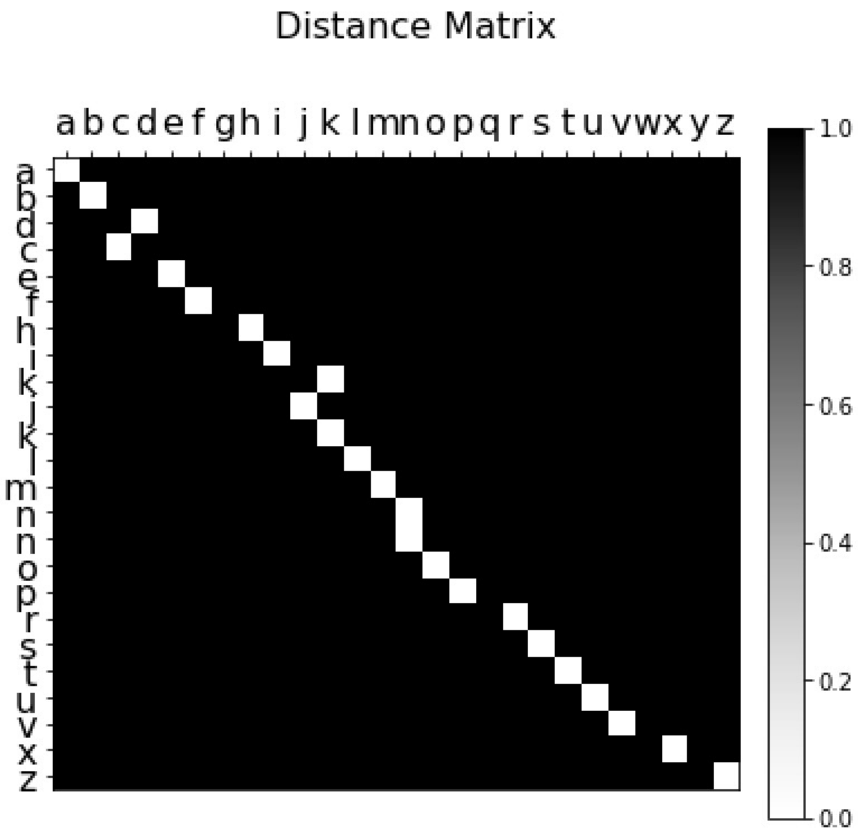
SCHNIDEIR $\xrightarrow[\text{delete}]{}$ **SHNIDEIR** $\xrightarrow[\text{delete}]{}$ **SNIDEIR** $\xrightarrow[\text{change}]{}$ **SNYDEIR** $\xrightarrow[\text{delete}]{}$ **SNYDER**

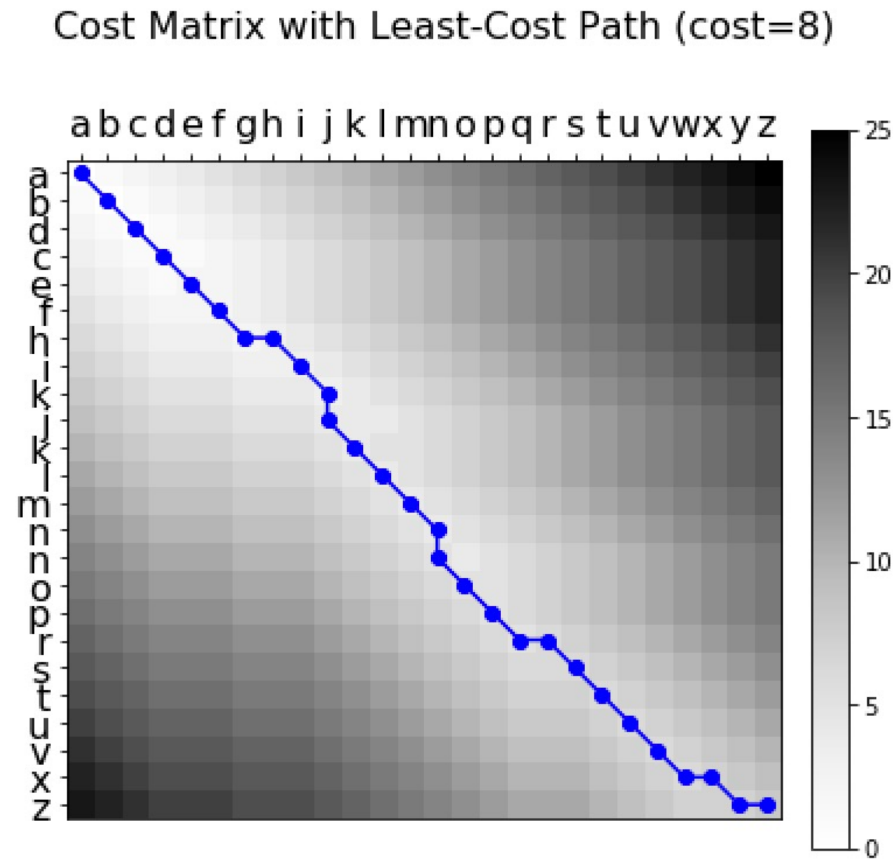




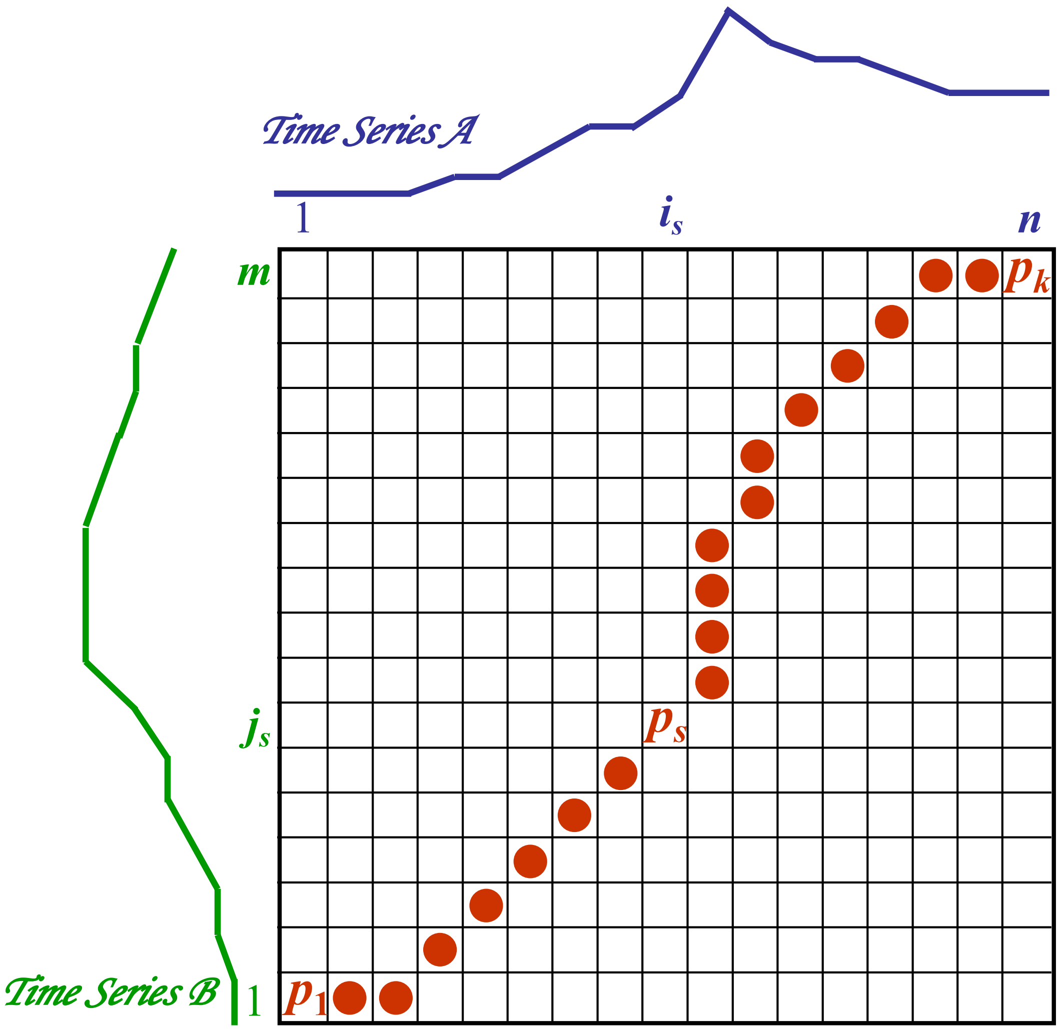
Music Synchronization: Audio-Audio

Finally, while creating the Cost Matrix, keep track of the minimum path from the upper left to the lower right corner:





When applied to two Time Series (e.g., audio signals) this technique is called Dynamic Time Warping.



To find the *best alignment* between $\mathcal{A}$ and $\mathcal{B}$ one needs to find the path through the grid

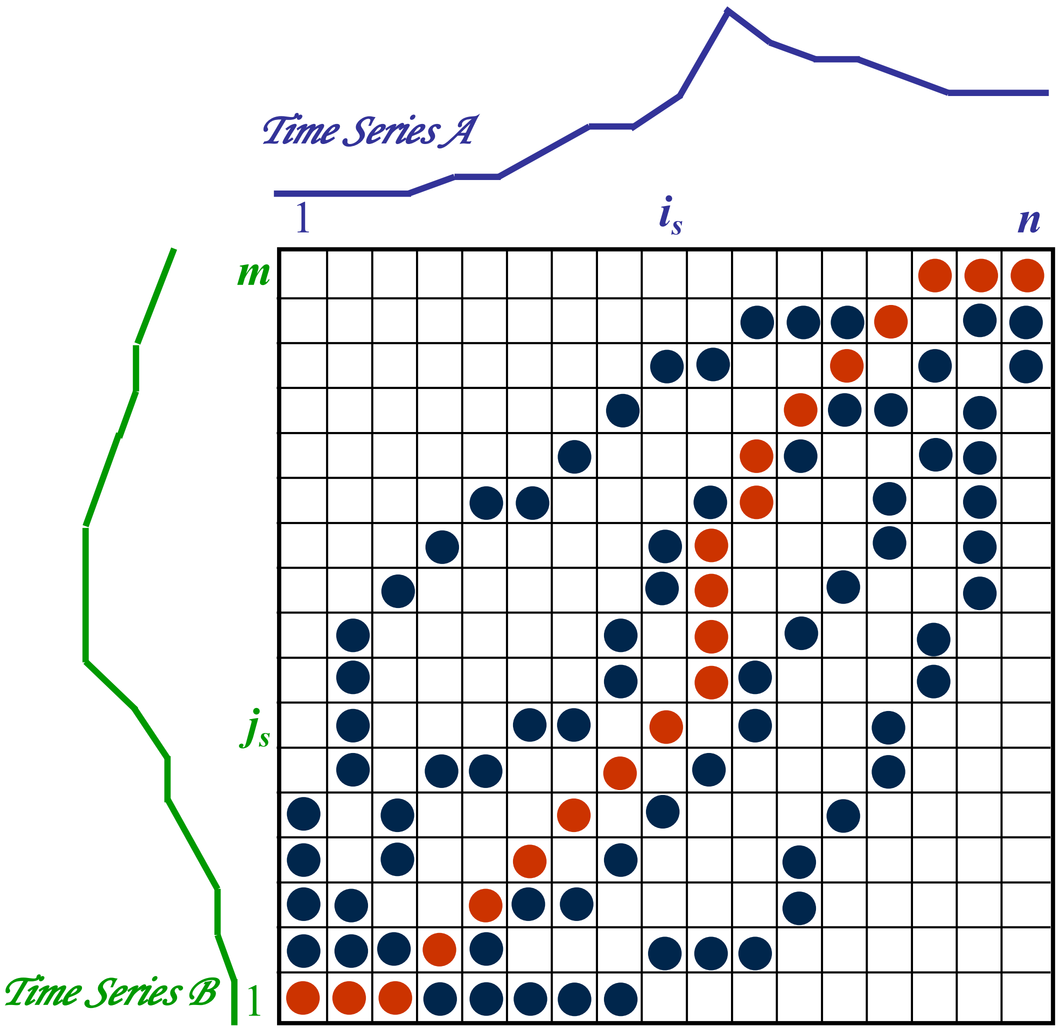
$$P = p_1, \ldots, p_s, \ldots, p_k$$

$$p_s = (i_s, j_s)$$

which *minimizes* the total distance between them.

P is called a *warping function*.

Optimisations to the DTW Algorithm



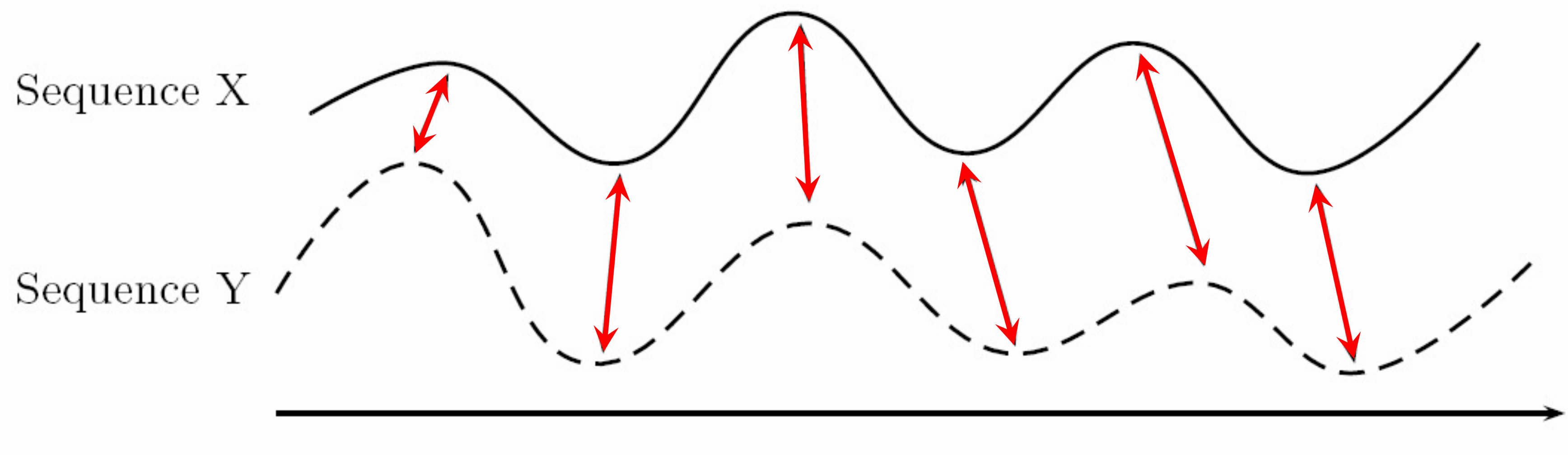
The number of possible warping paths through the grid is exponentially explosive!

reduction of the search space

Restrictions on the warping function:

- monotonicity
- continuity
- boundary conditions
- warping window
- slope constraint.

The result is an alignment between the two signals which can then be used for score alignment, etc.

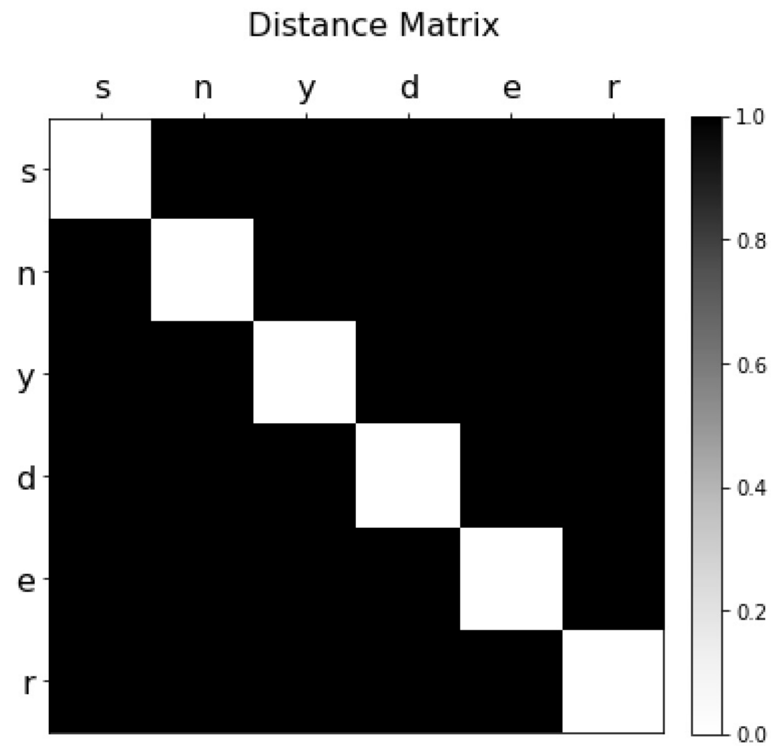


Or one sequence can be time shifted (using a vocoder) so that it exactly matches the timing of the other.

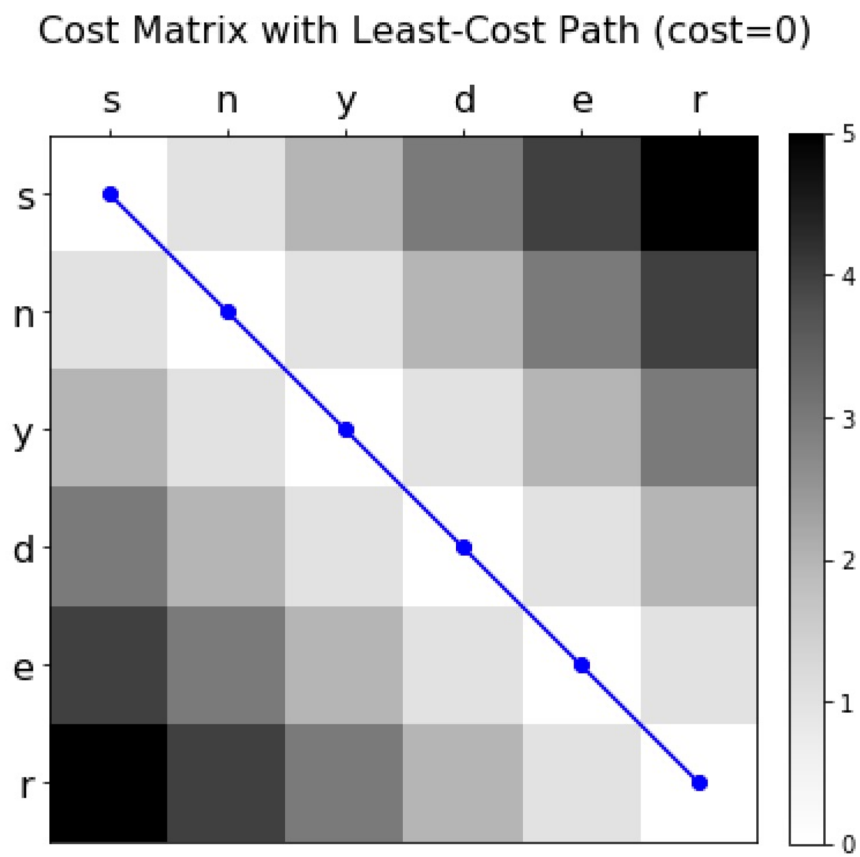
Music Structure Analysis

In approximate string matching, it is interesting to try the self-similarity of a string of symbols…..

SNYDER SNYDER



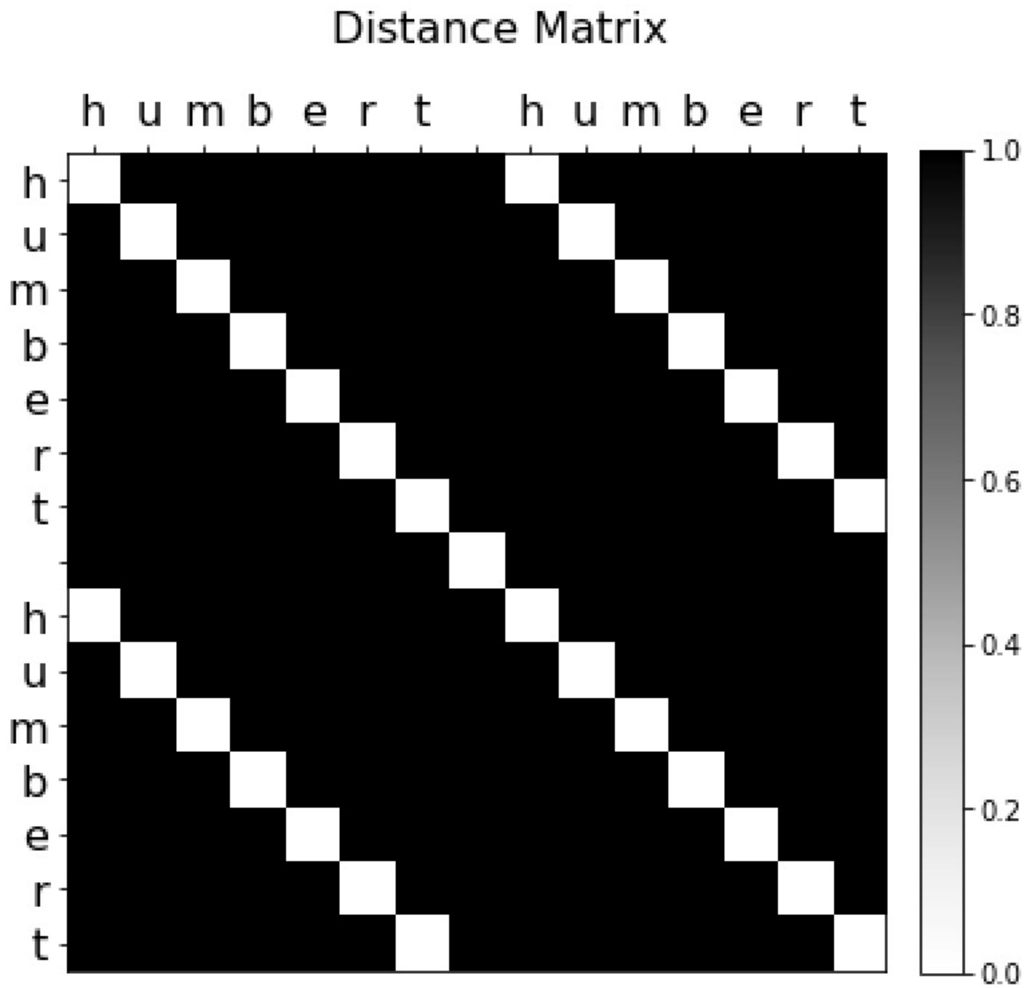
0.0	1.0	1.0	1.0	1.0	1.0
1.0	0.0	1.0	1.0	1.0	1.0
1.0	1.0	0.0	1.0	1.0	1.0
1.0	1.0	1.0	0.0	1.0	1.0
1.0	1.0	1.0	1.0	0.0	1.0
1.0	1.0	1.0	1.0	1.0	0.0



Music Synchronization: Audio-Audio

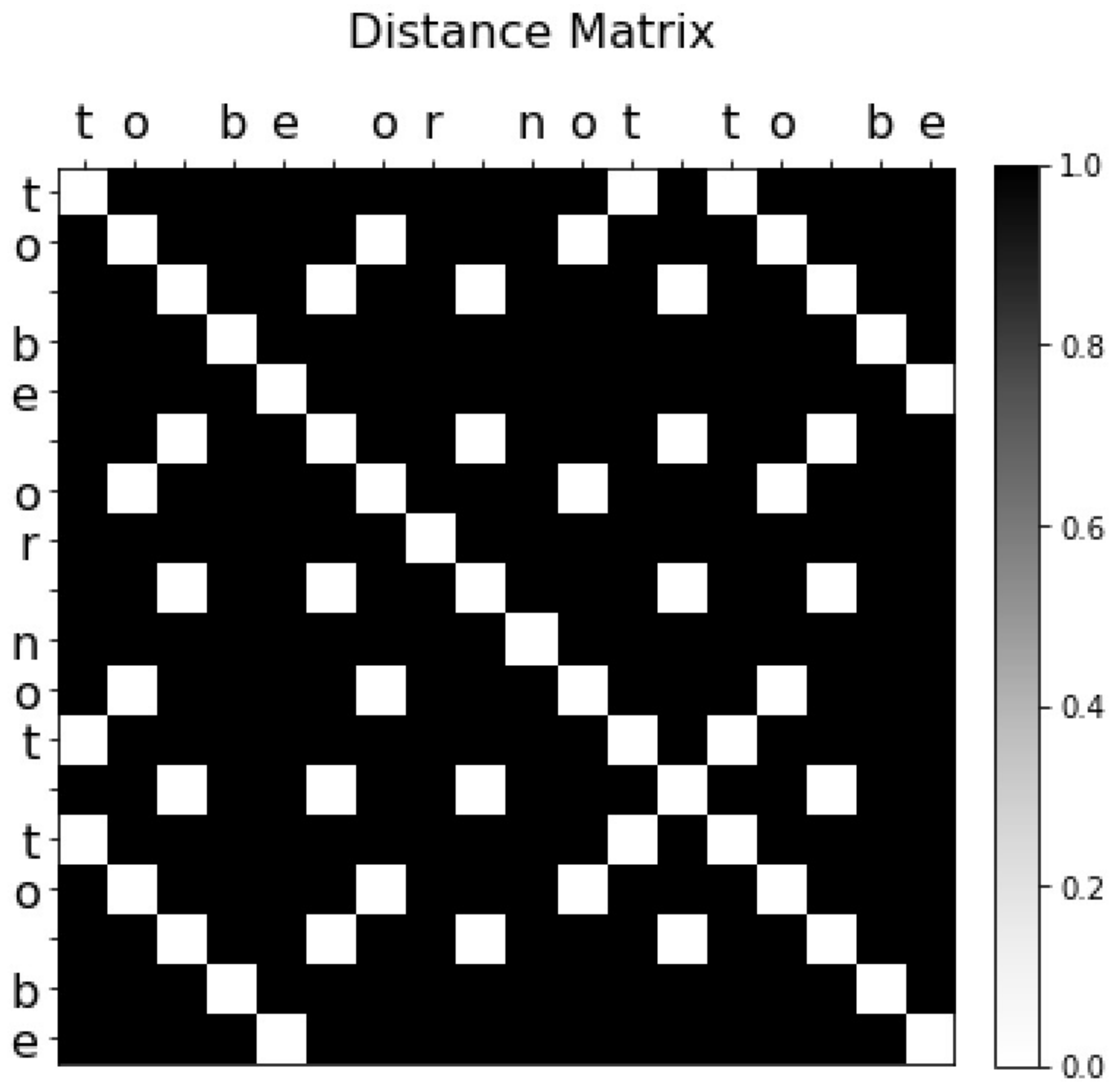
In approximate string matching, it is interesting to try the self-similarity of a string of symbols…..

HUMBERT HUMBERT



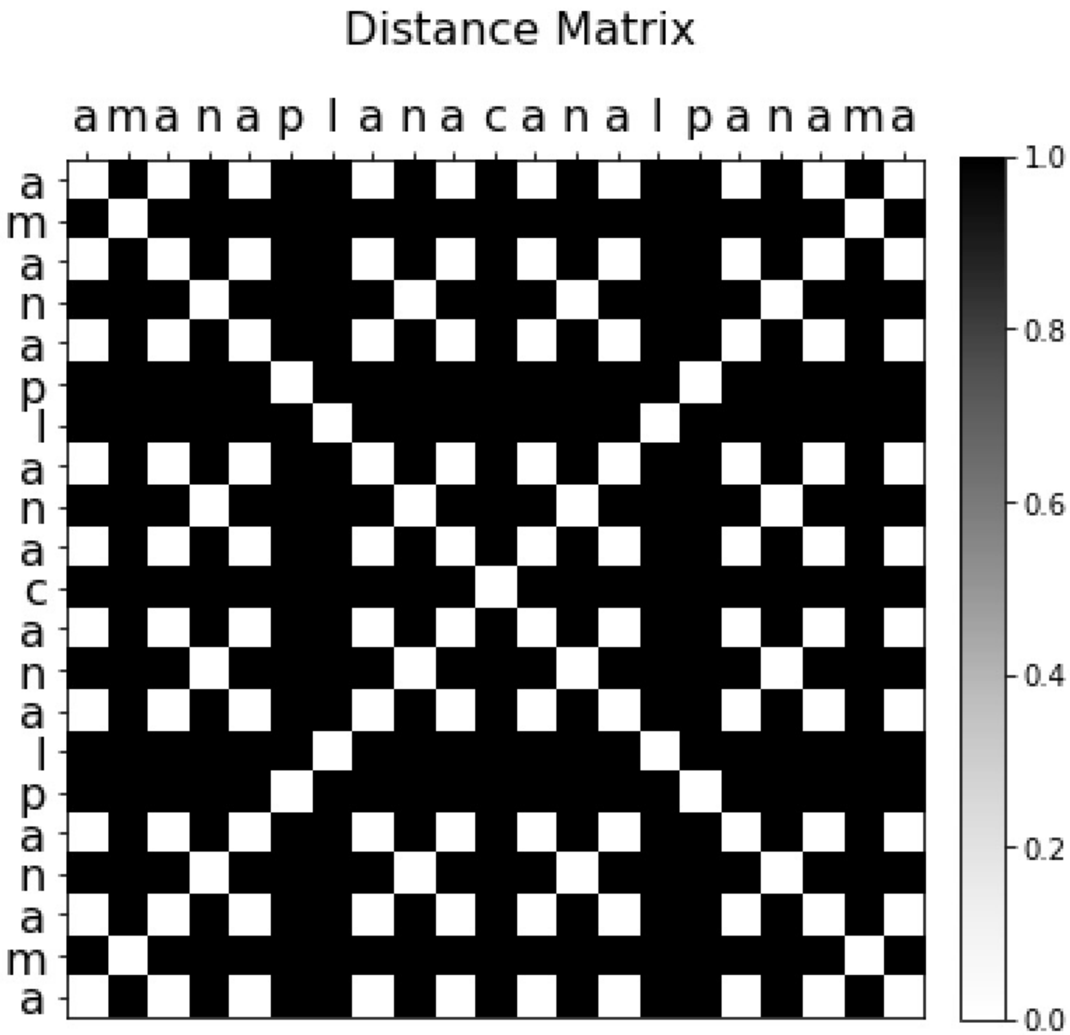
Music Synchronization: Audio-Audio

In approximate string matching, it is interesting to try the self-similarity of a string of symbols…..



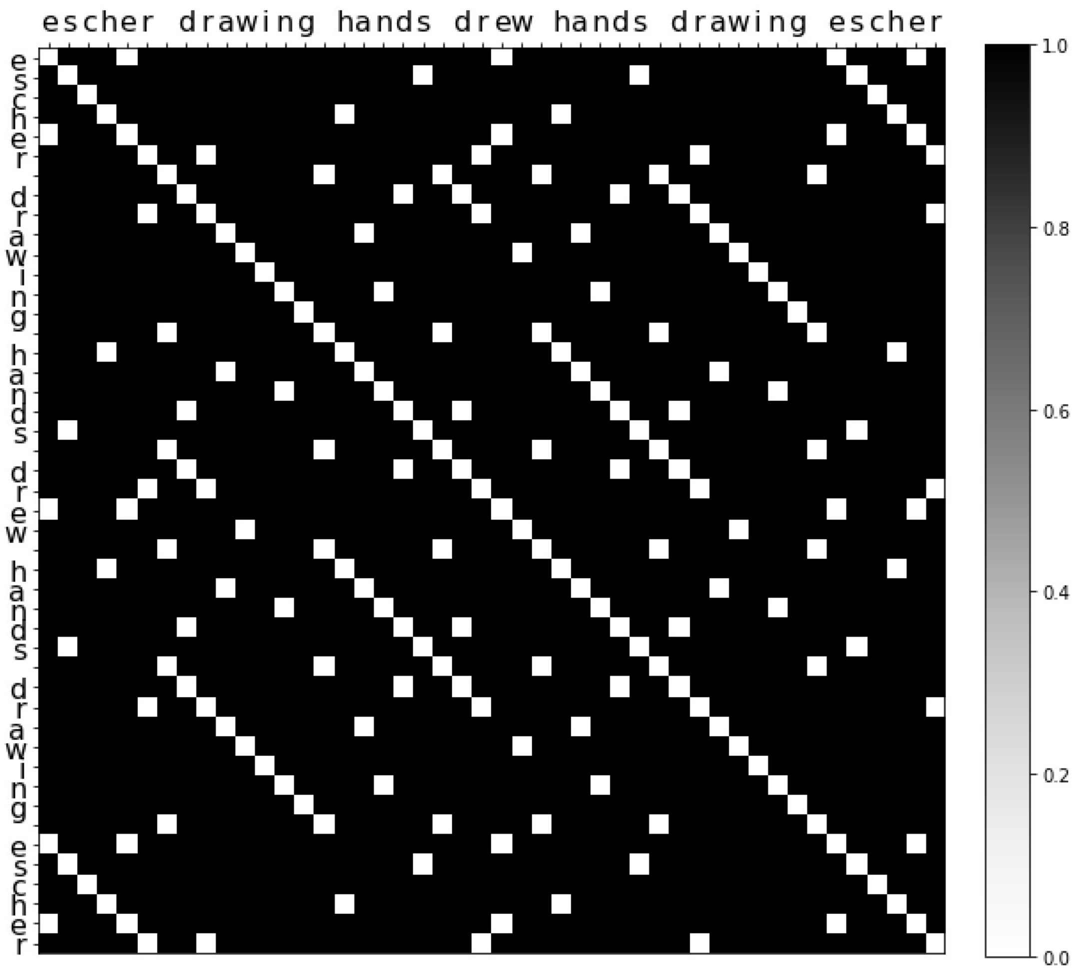
Music Synchronization: Audio-Audio

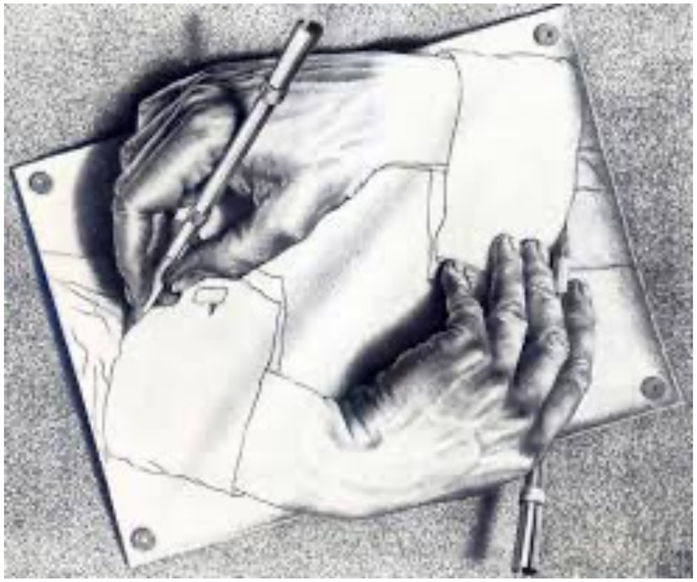
In approximate string matching, it is interesting to try the self-similarity of a string of symbols…..



Music Synchronization: Audio-Audio

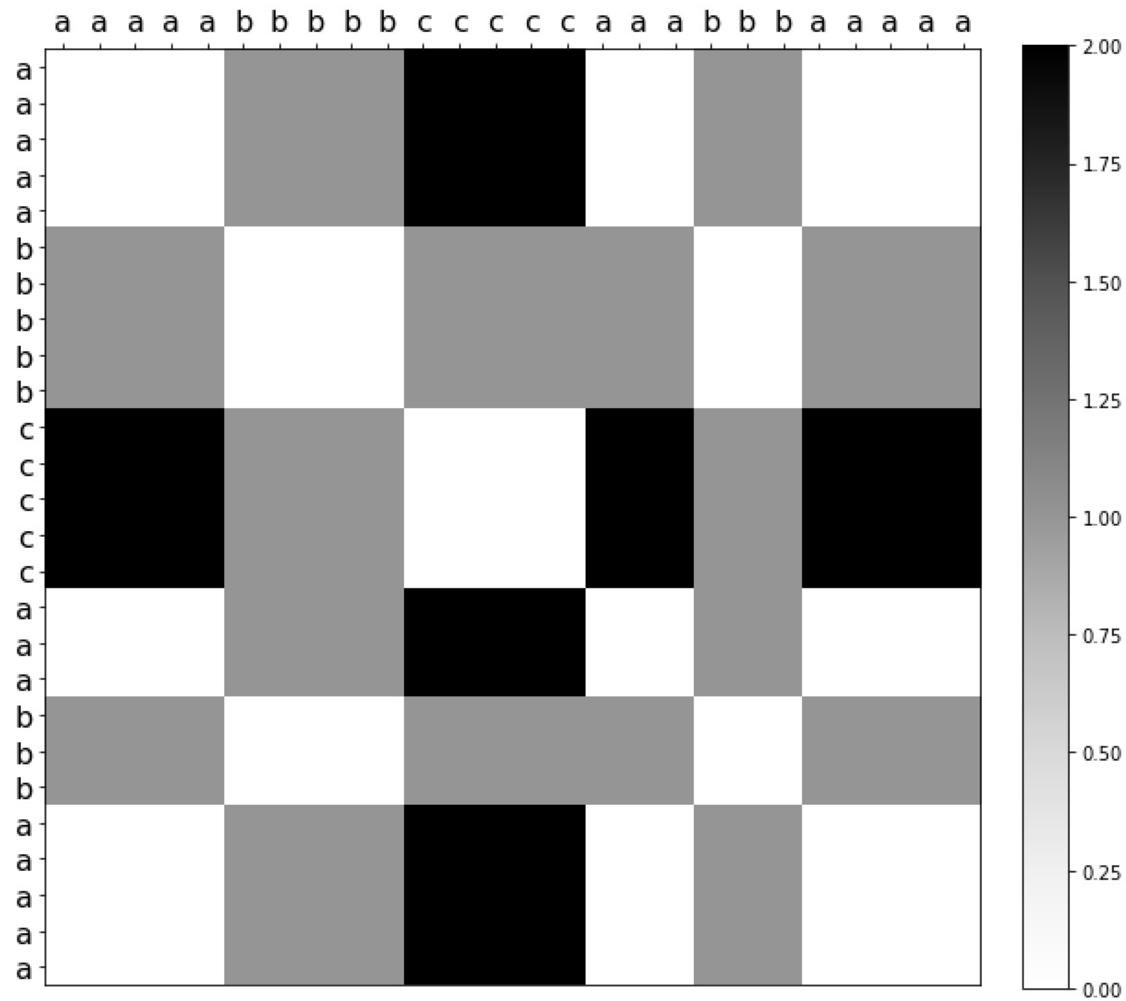
In approximate string matching, it is interesting to try the self-similarity of a string of symbols…..





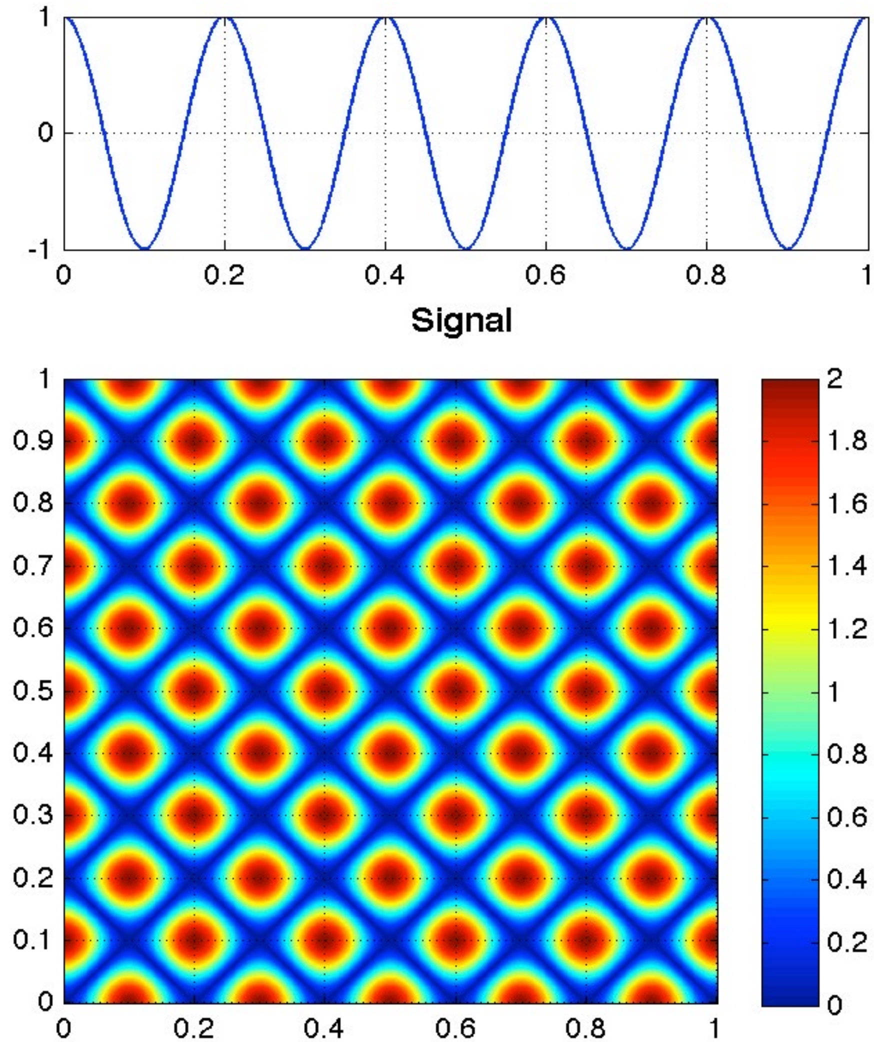
Music Synchronization: Audio-Audio

In approximate string matching, it is interesting to try the self-similarity of a string of symbols…..

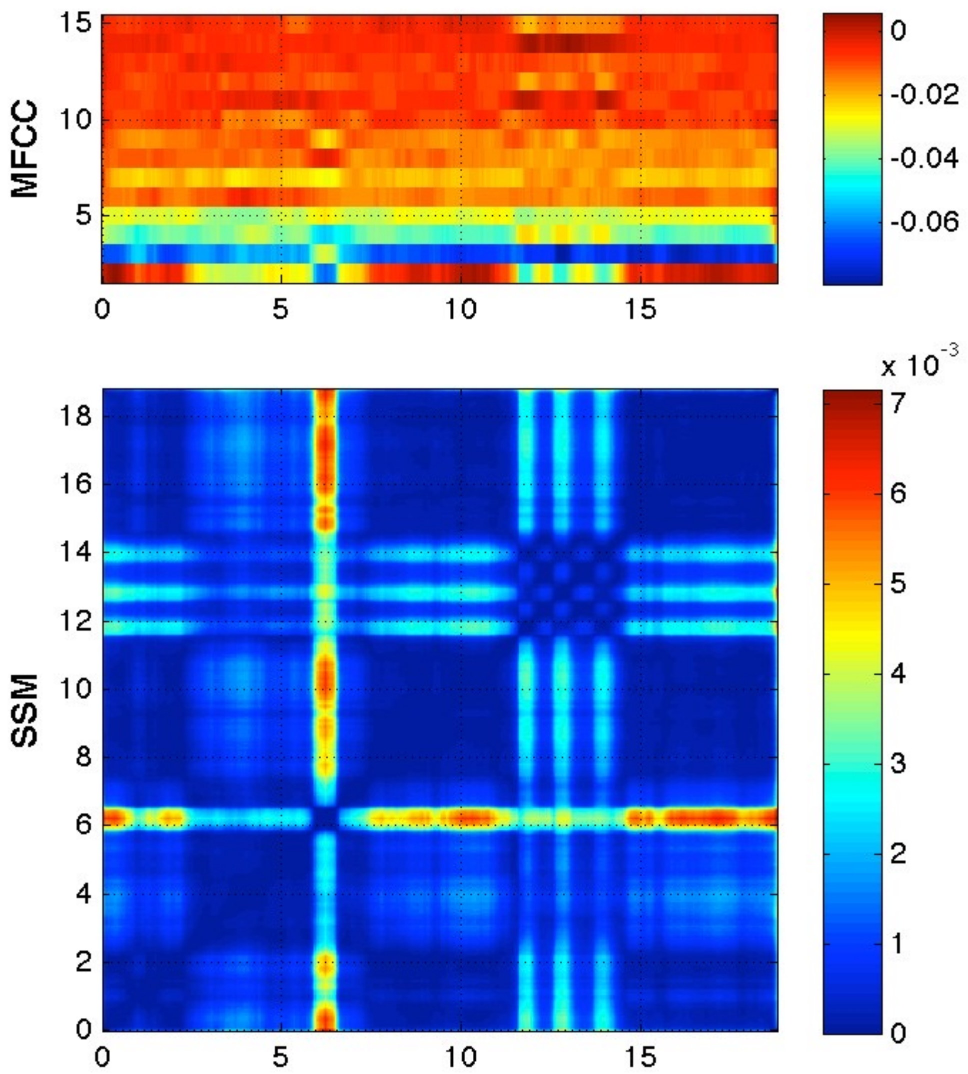


Self-similarity matrix for Audio Signals

- Vertical and horizontal axes represent time
- symmetric similarity function = symmetric matrix of distance measures
- Main diagonal: closer/ most similar values
- similar subsequences (repetitions) -> diagonal stripes in the plot



Self-similarity matrix



Identifying structure from audio

- Arthur G. Lintgen: able to identify unlabeled recorded orchestral works by observing the spacing and patterns of grooves in an LP

Lintgen setting his sights on the patterns in some new discs

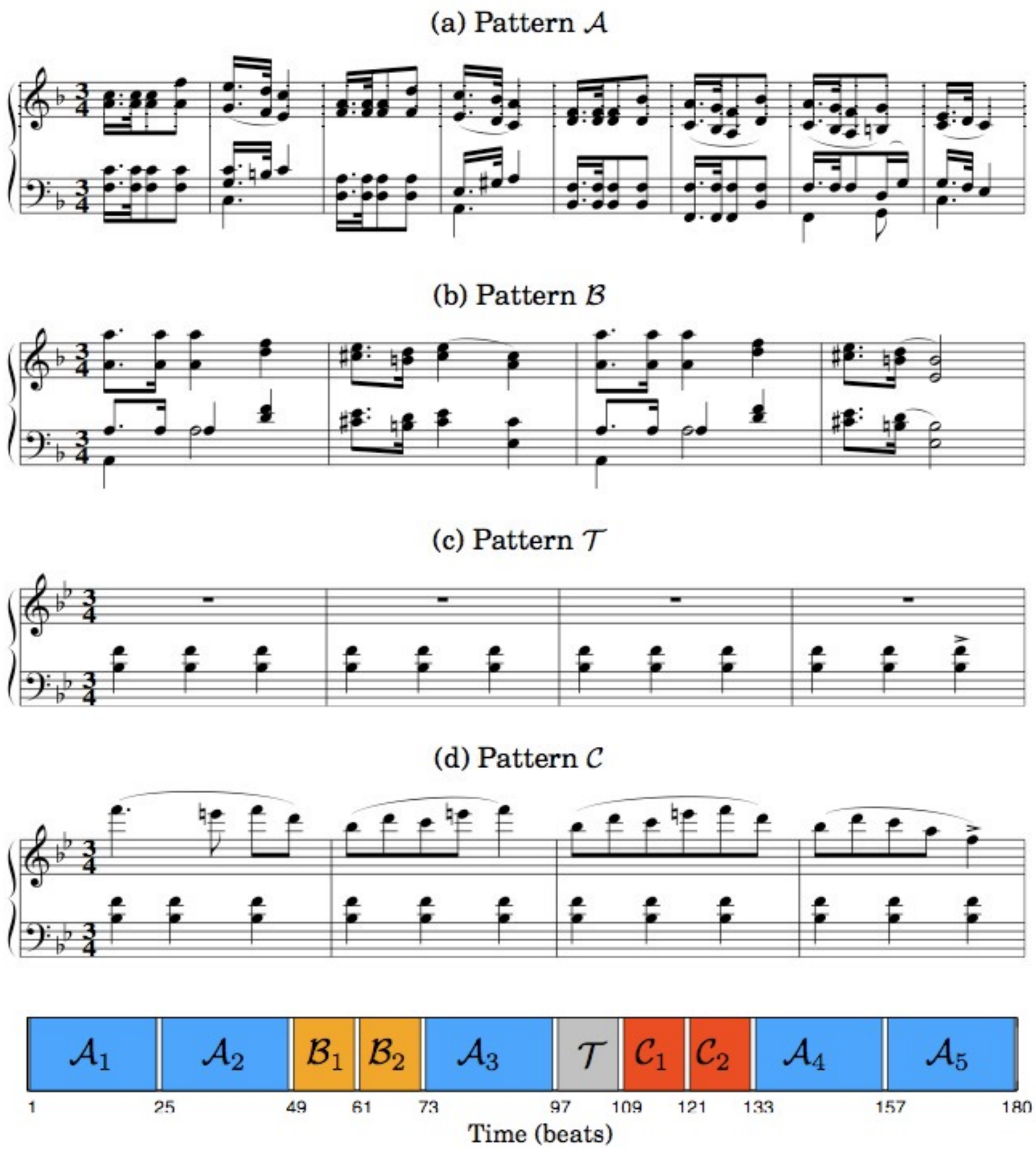
- Inspired J. Foote (ISMIR, 2000) to develop a MIR system based on structural similarity

Musical Form

- Units can be assigned letters (A, B, C) or functional names (intro, verse, chorus, bridge, etc)
- Strophic: repeats the same section, e.g. AA…
- Binary: alternates two sections, which are often repeated, e.g. ABAB or AABB
- Ternary: third section is often a variation of the first, e.g. AABA, AABA', AA' BA'
- Arch: symmetric, repetition of sections around a center, e.g. ABCBA
- Rondo: main theme is alternated with sub-themes, e.g. ABACADA…..
- Variations: theme plus variations, e.g. $AA^{i}A^{ii}AA^{iii}$
- Sonata: complex developmental form including the exposition, development and recapitulation of a given theme(s).

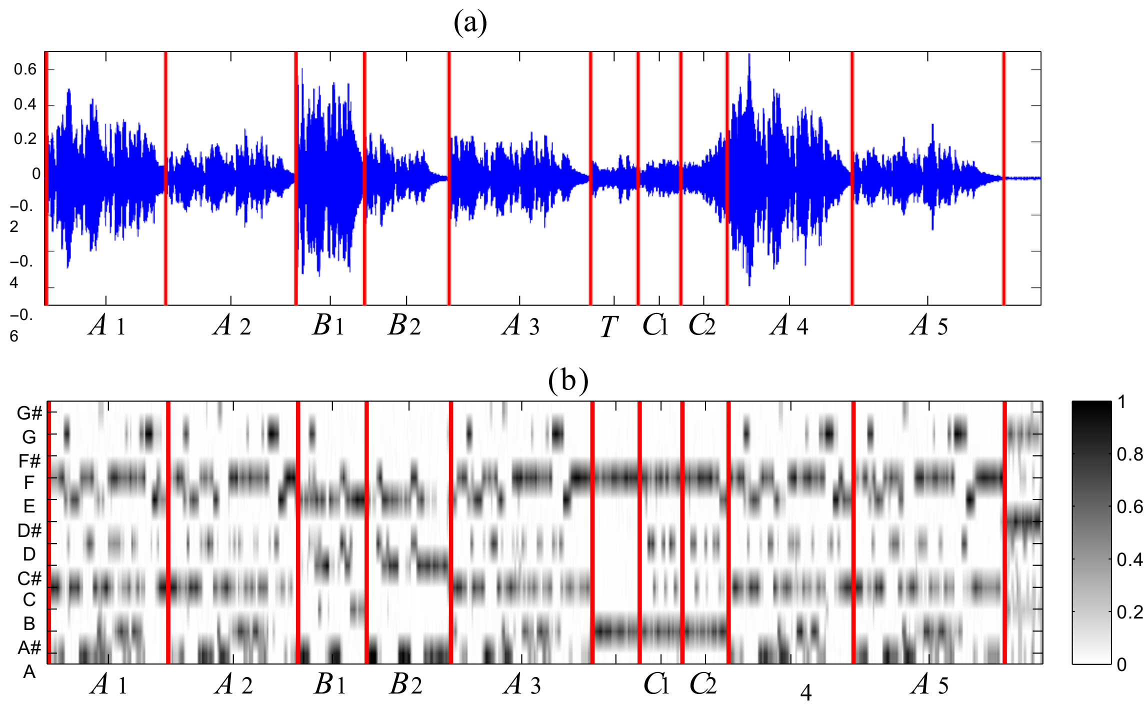
Repetition

- Musical form is often defined by the amount of repetition across sectional units.
- Repetition is central to music (in harmony, melody, rhythm, etc).
- Significant variations are often found between repeated parts.



Repetition

- The information necessary to characterize repetitions is encoded in the feature vectors (e.g., chroma, spectrum, etc.)

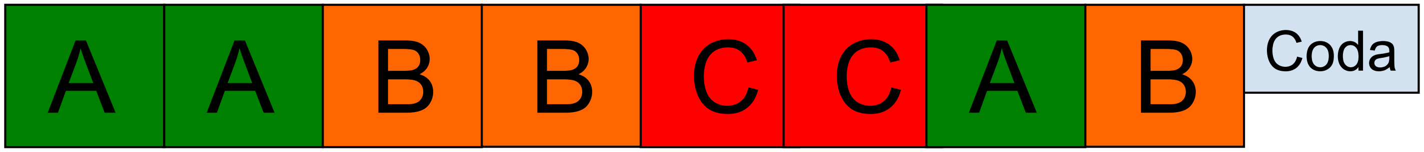


Audio Structure Analysis

Given: CD recording

Goal: Automatic extraction of the repetitive structure
(or of the musical form)

Example: Brahms Hungarian Dance No. 5

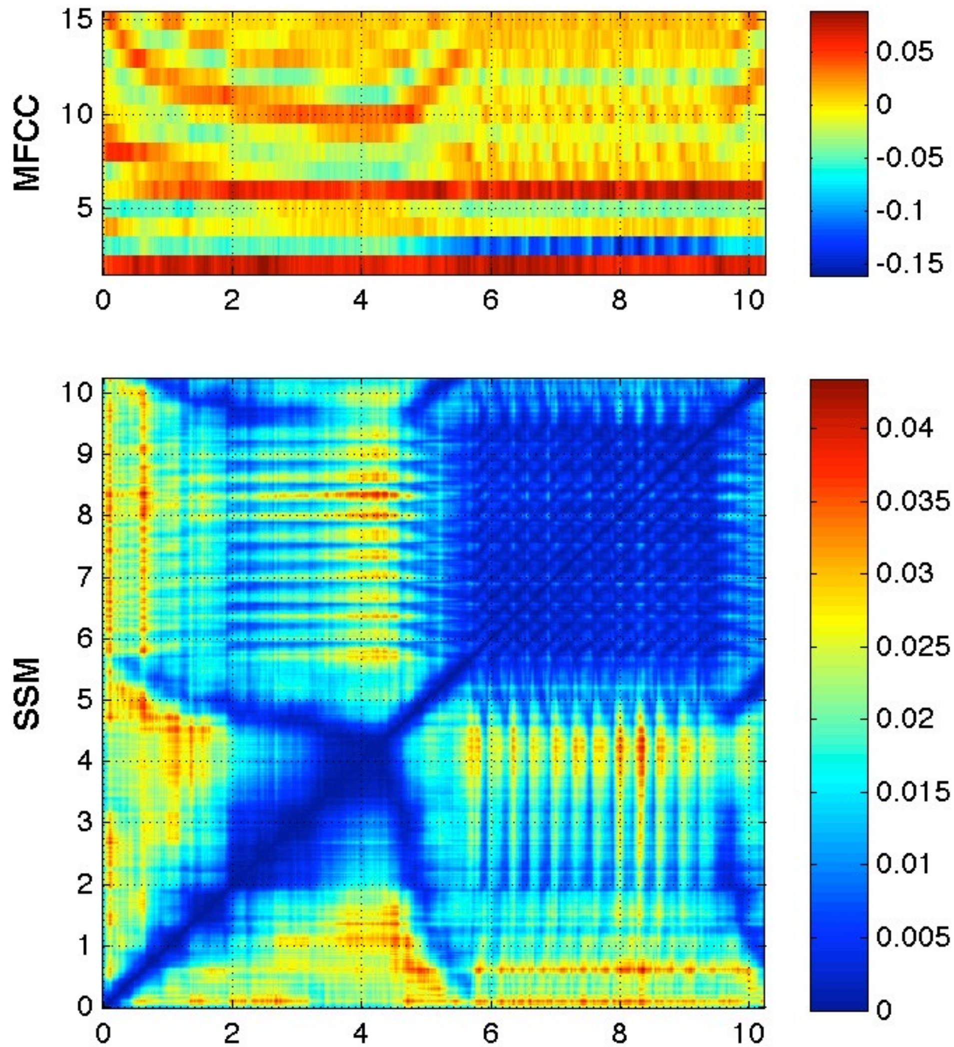


Basic Procedure

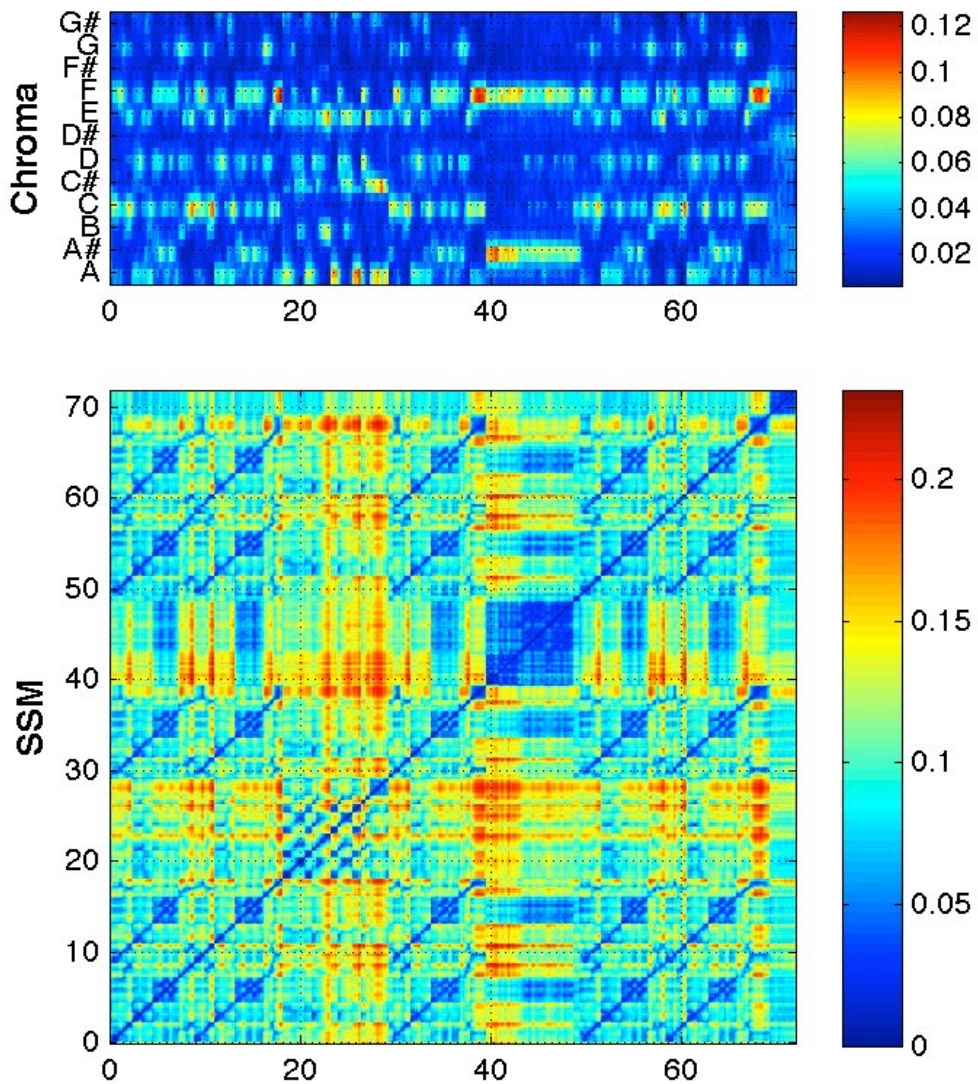
- Extract audio feature vectors (e.g., spectrograph, mel spectrograph, chromagraph)
- Cost measure and cost matrix

  self-similarity matrix
- Path extraction (pairwise similarity of segments)
- Global structure (clustering, grouping)

Self-similarity matrix



Self-similarity matrix



Let's look at a similarity matrix and hear the piece of music to see how it represents the structure....

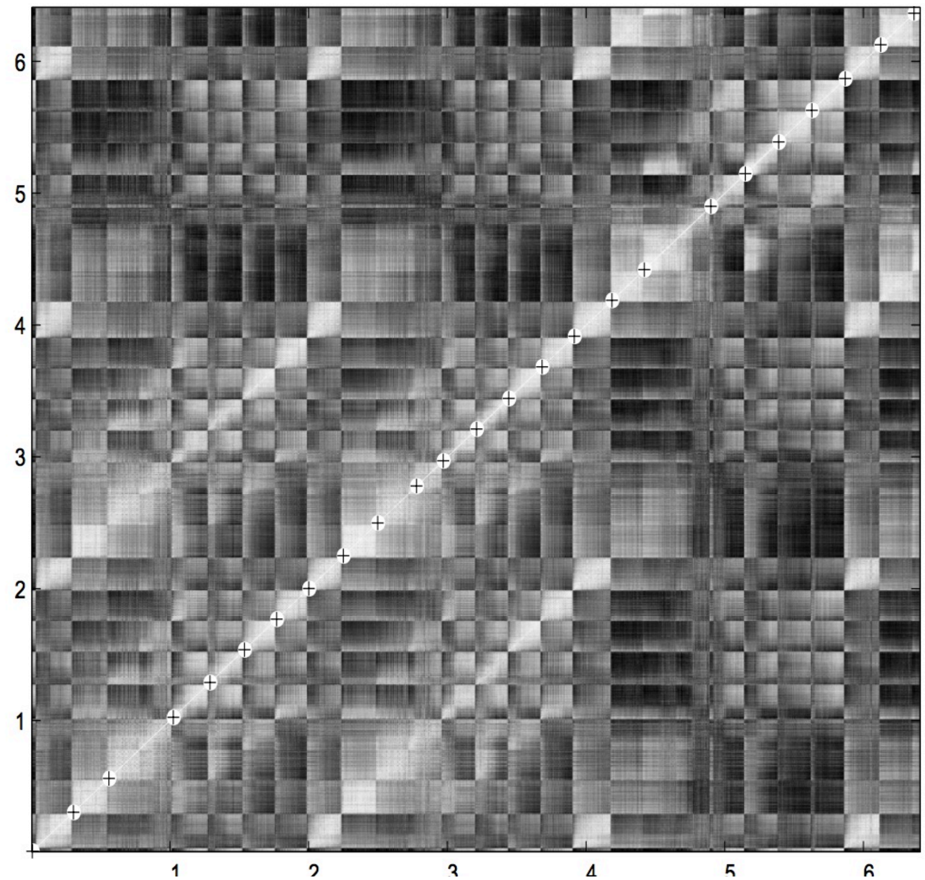
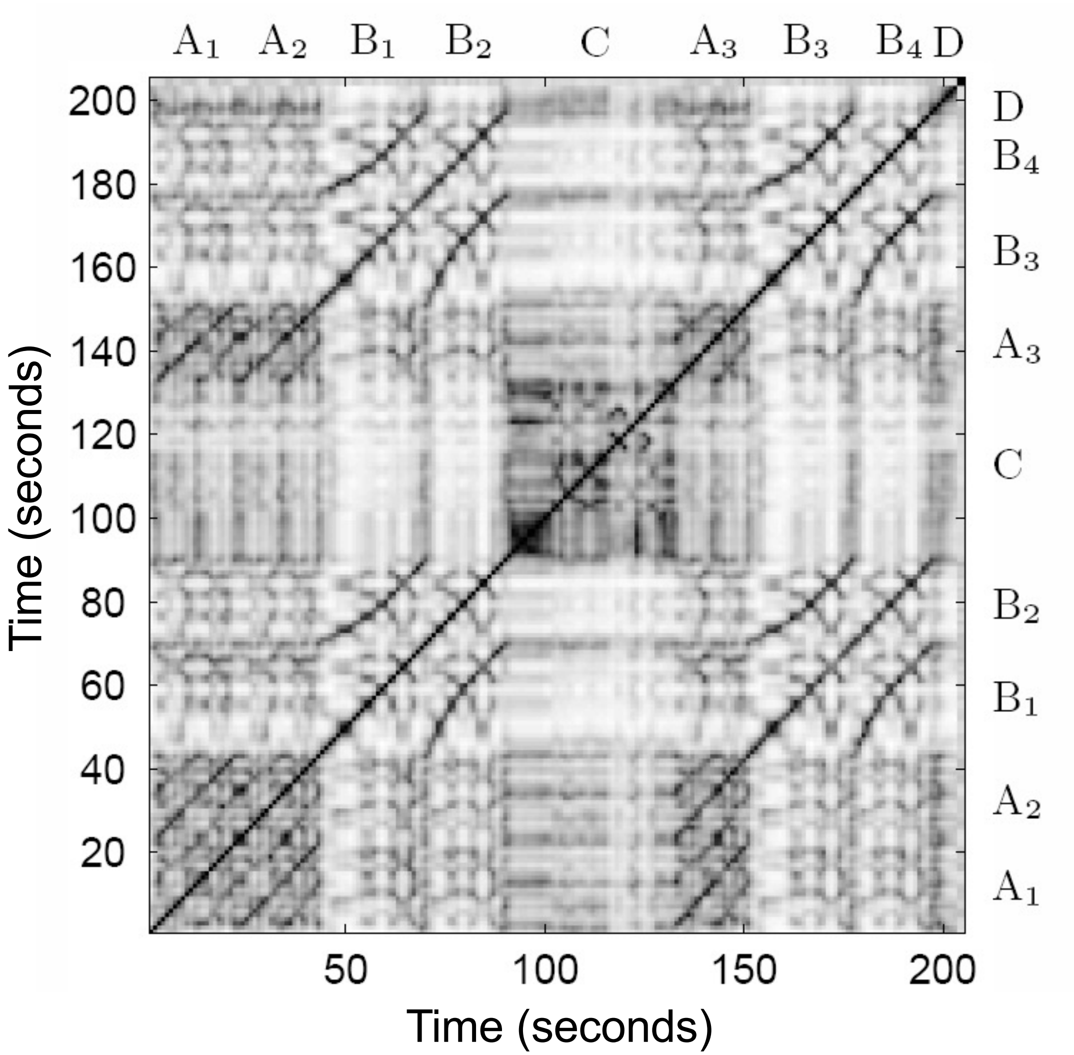


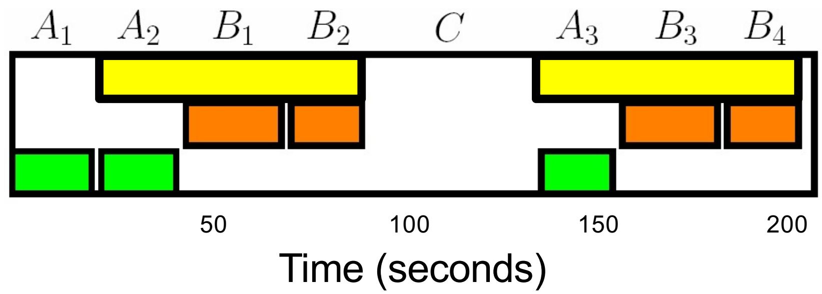
Figure 1. First bars of Bach's *Prelude No. 1 in C Major*, BVW 846, from *The Well-Tempered Clavier*

Basic Procedure

Self-similarity matrix

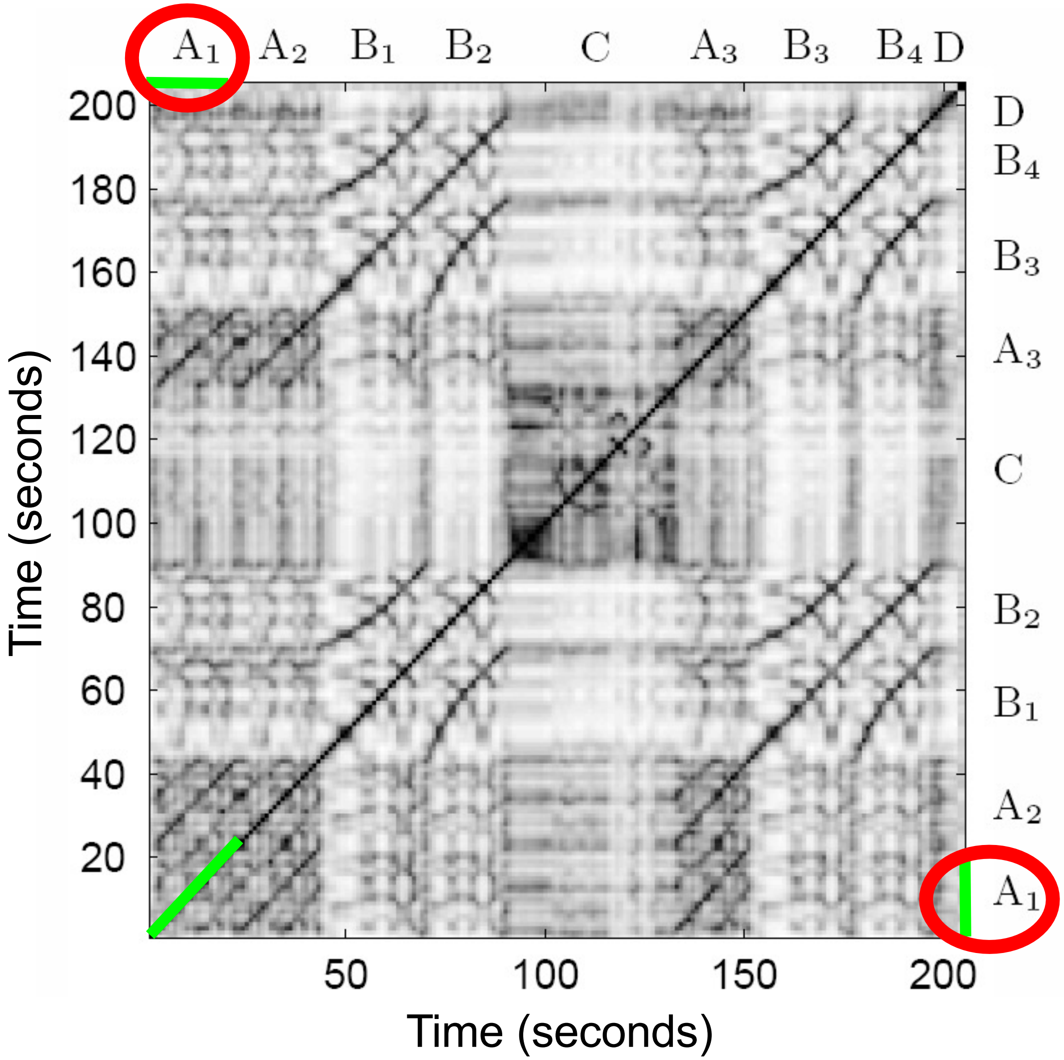


Similarity structure

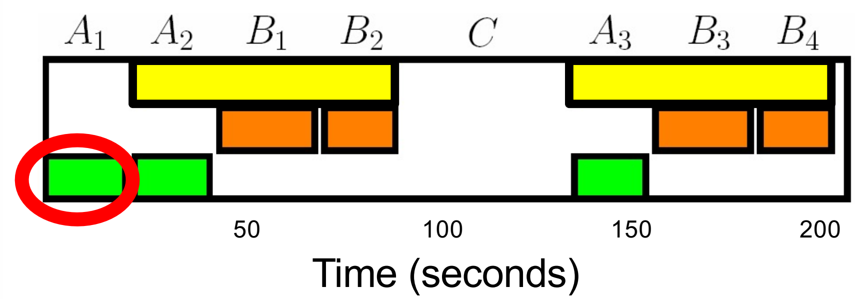


Basic Procedure

Self-similarity matrix

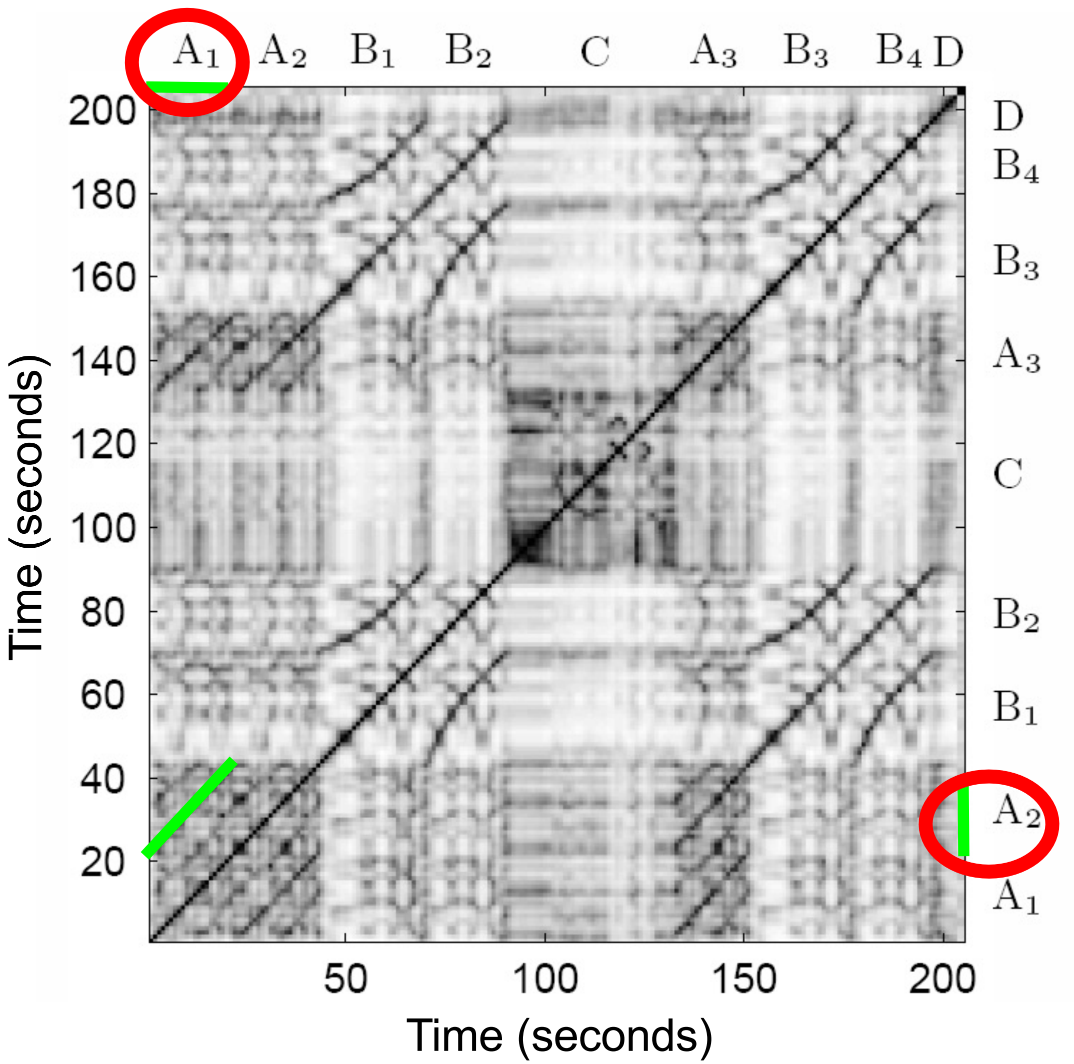


Similarity structure

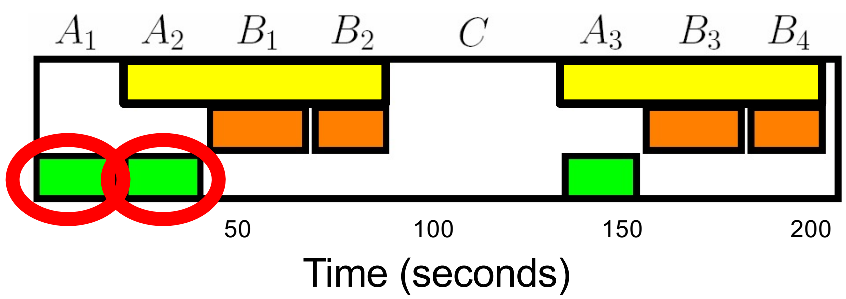


Basic Procedure

Self-similarity matrix

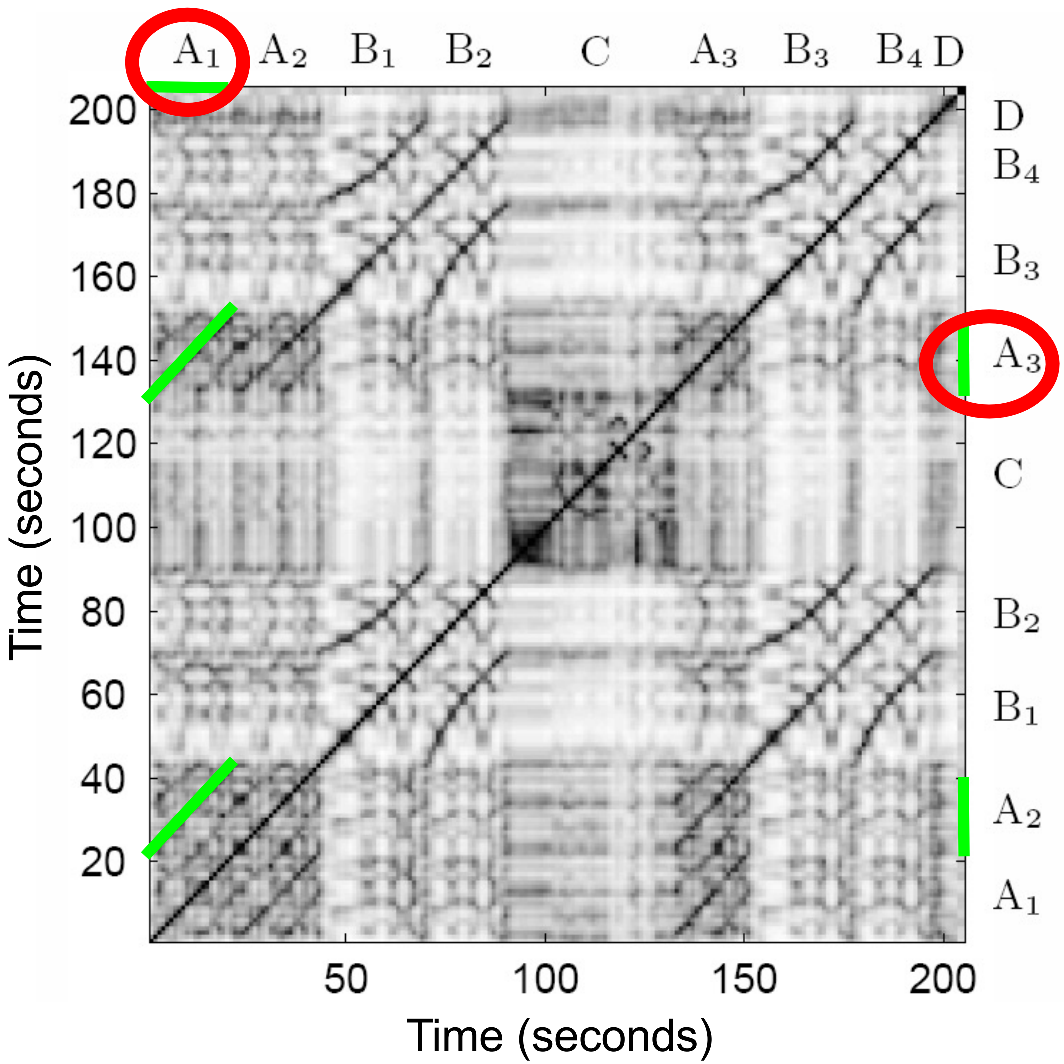


Similarity structure

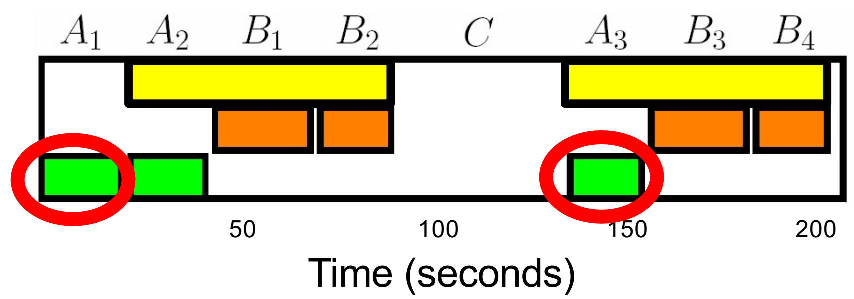


Basic Procedure

Self-similarity matrix

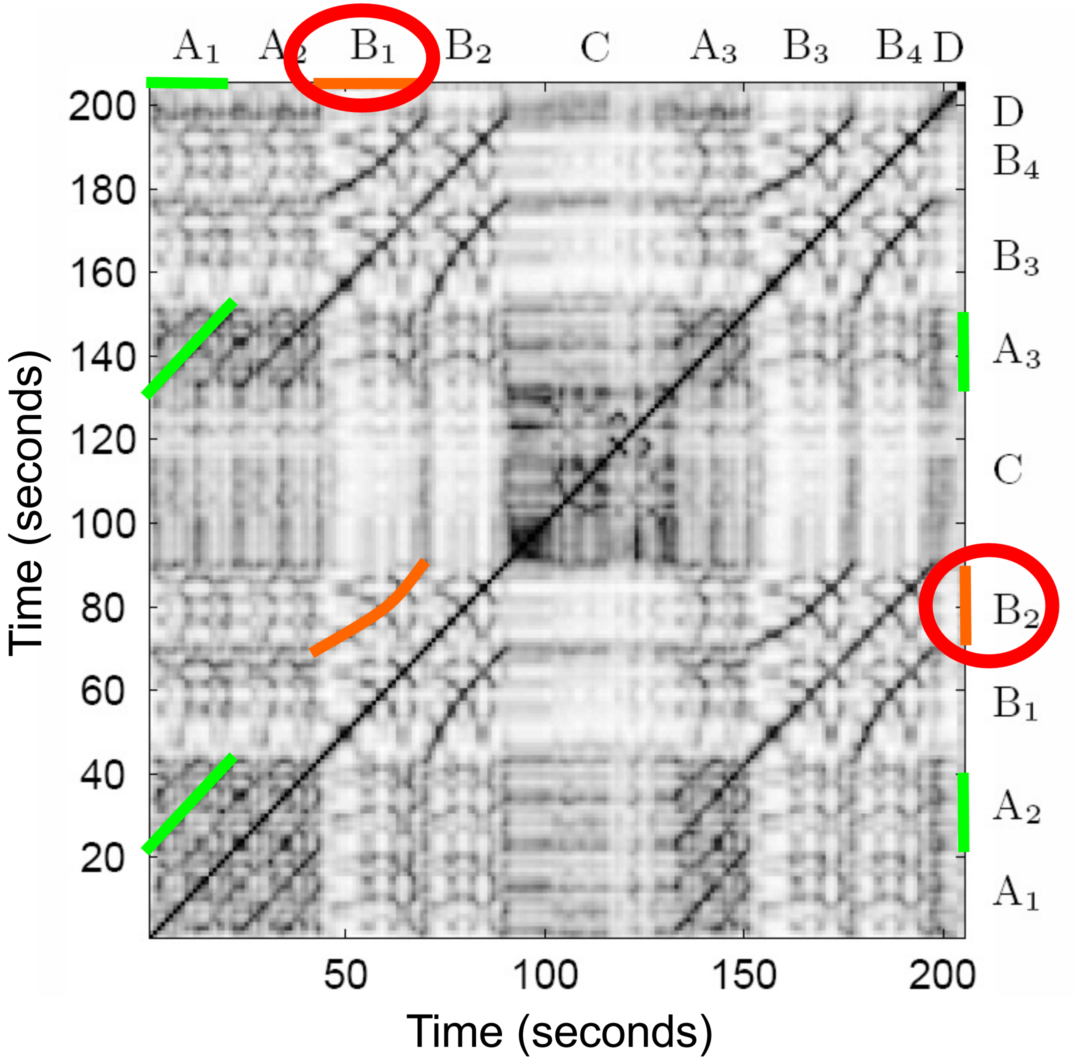


Similarity structure

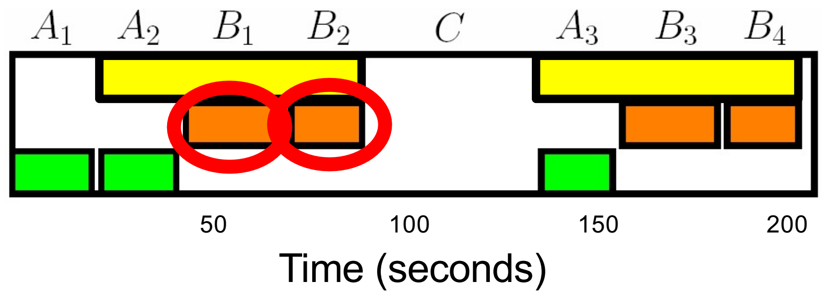


Basic Procedure

Self-similarity matrix

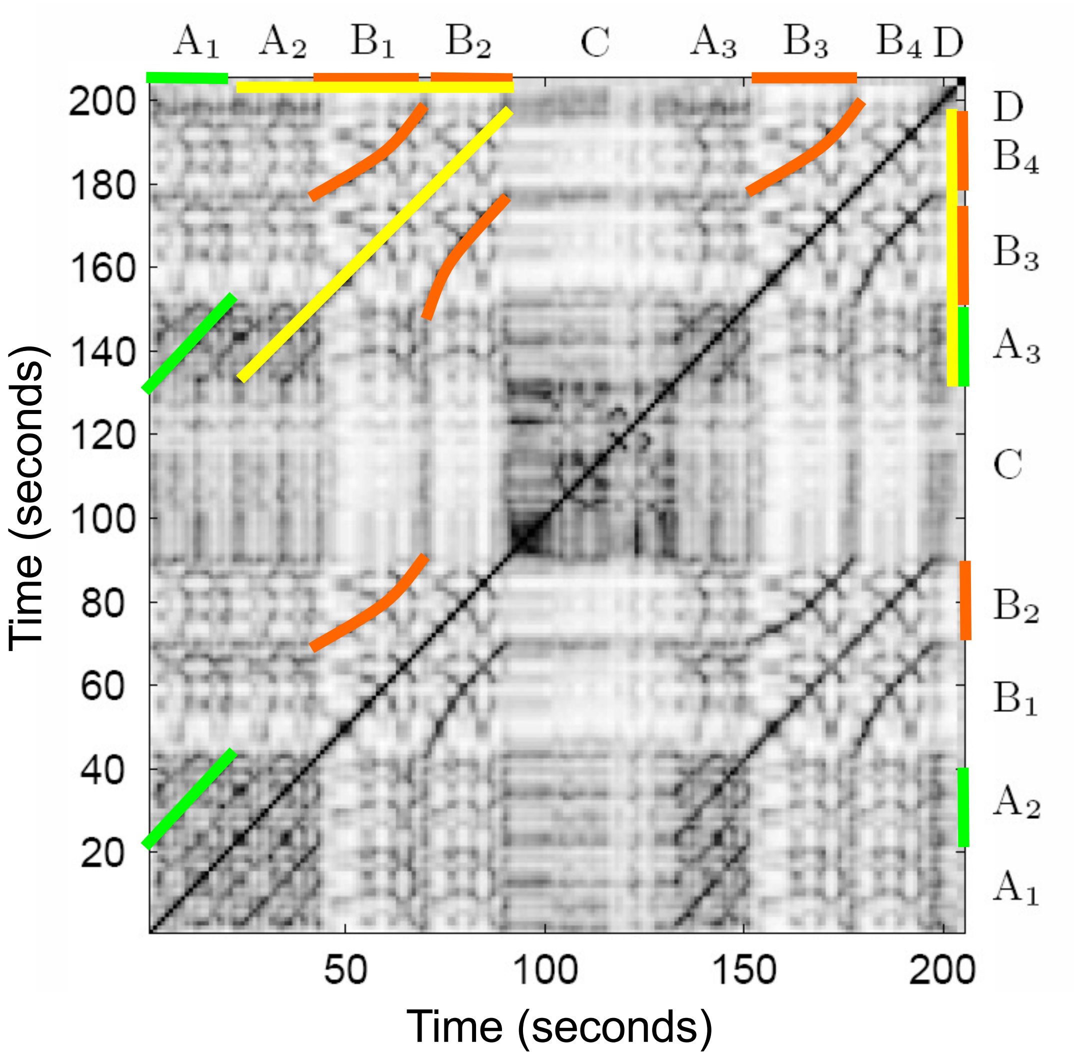


Similarity structure

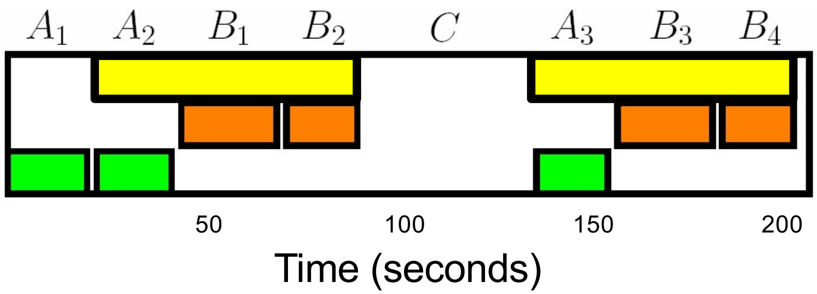


Basic Procedure

Self-similarity matrix



Similarity structure



High-Resolution Music Synchronization

- Normalized chroma features
  → robust to changes in instrumentation and dynamics
  → robust synchronization of reasonable overall quality

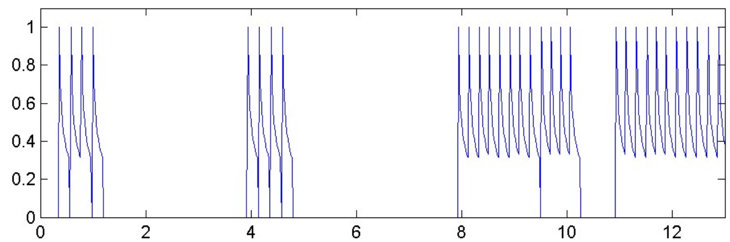
- Drawback: low temporal alignment accuracy

- Idea: Integration of note onset information

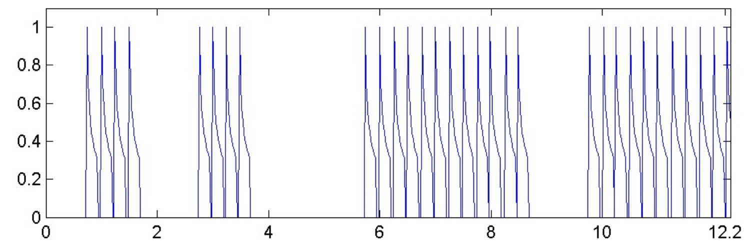
High-Resolution Music Synchronization

Cost matrix windows are based on based on onset intervals, not uniformly spaced!

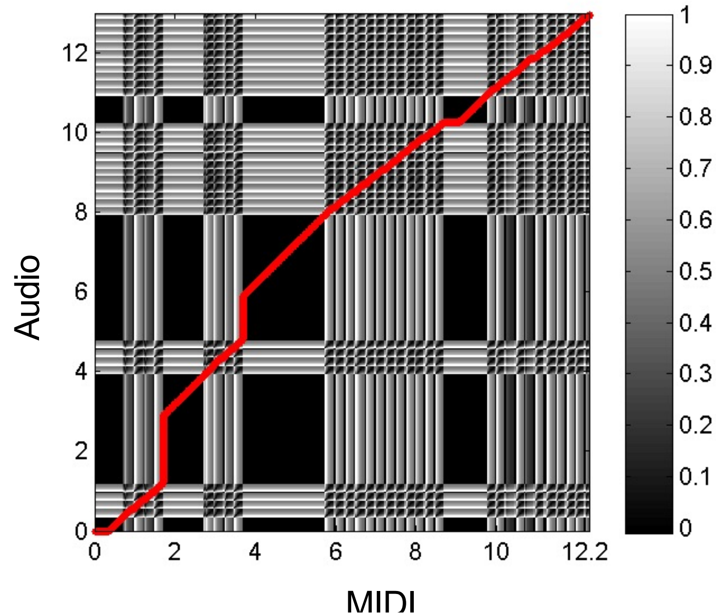
Audio



MIDI



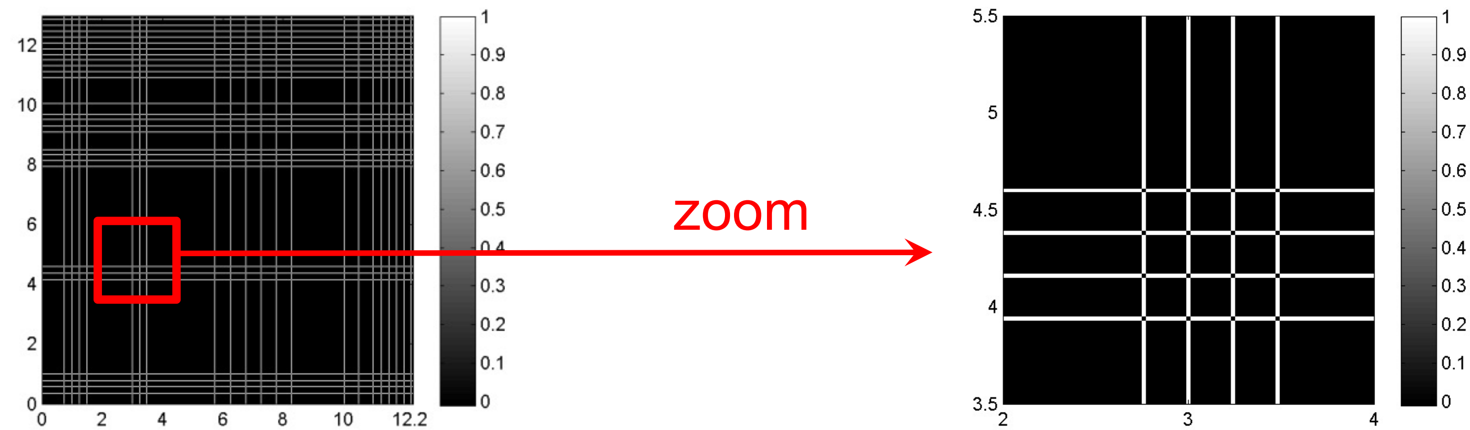
Cost matrix



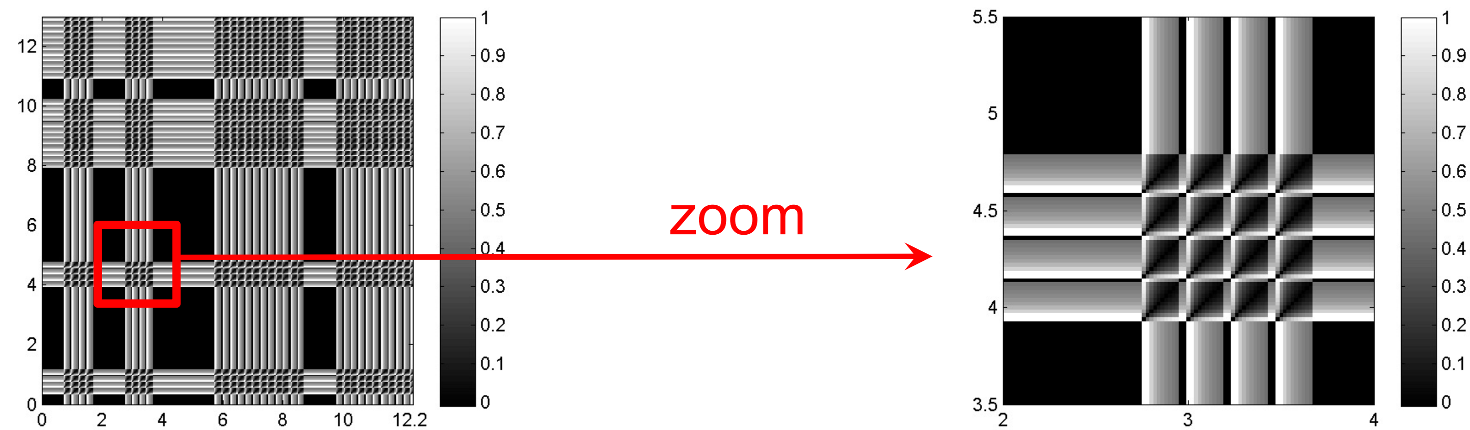
Warping path based on onset information

High-Resolution Music Synchronization

Impulses



Decaying impulses



Music Segmentation Analysis

- Music segmentation
  - pitch content (e.g., melody, harmony)
  - music texture (e.g., timbre, instruments)
  - rhythm

  - How to find the musical “sections” of the piece?

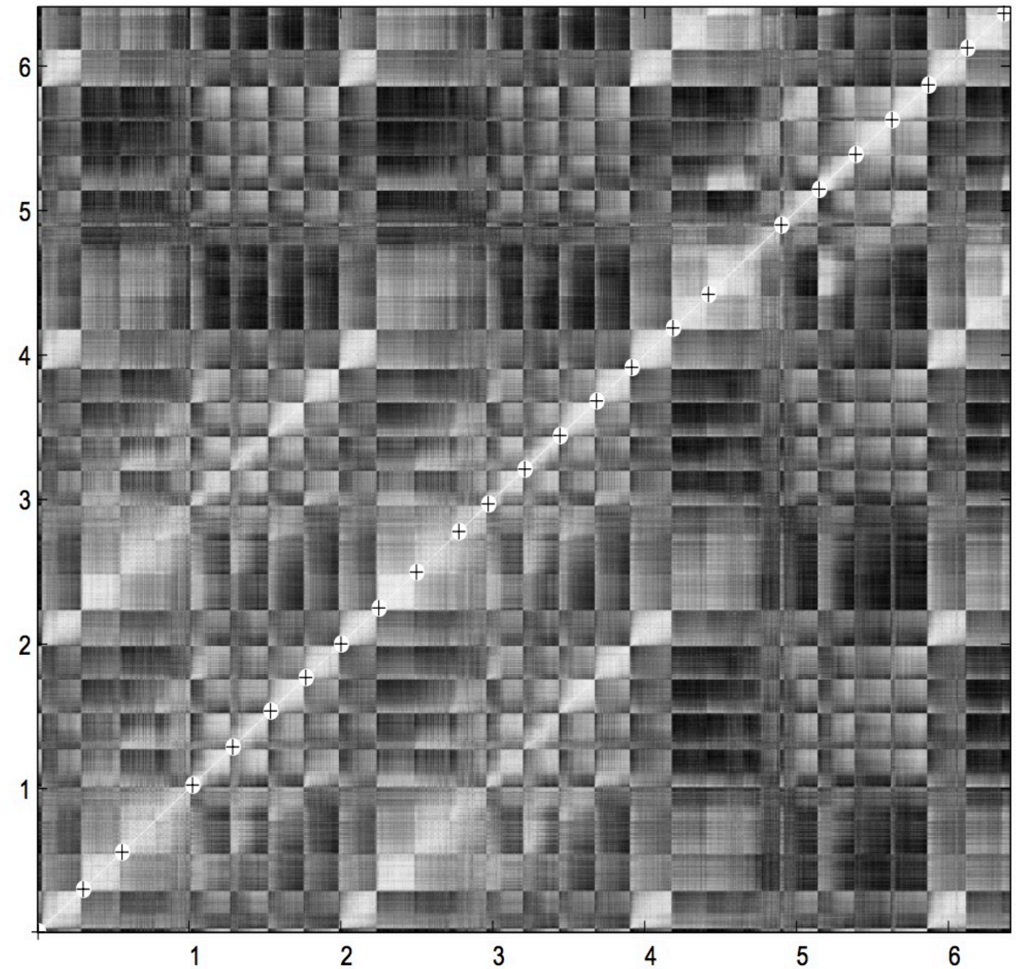
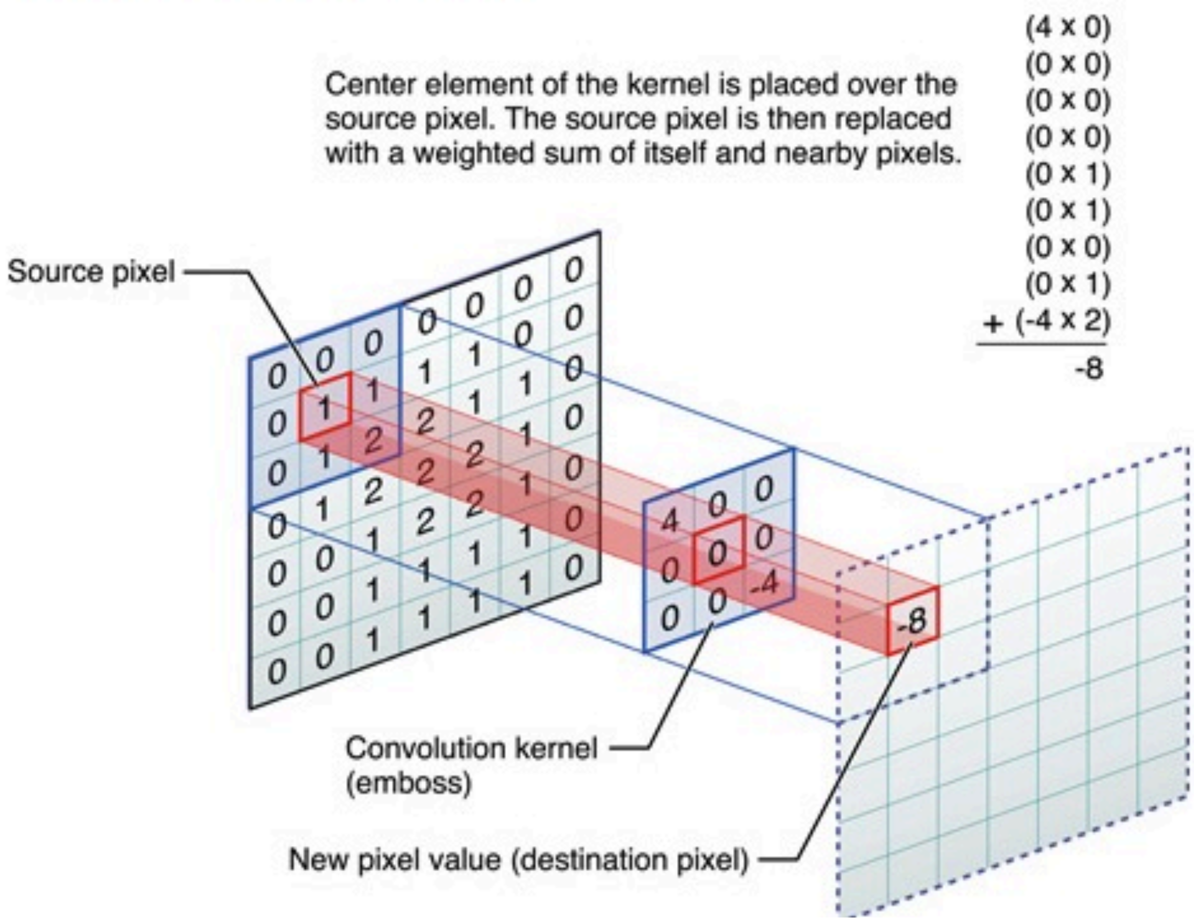


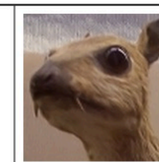
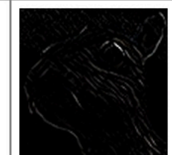
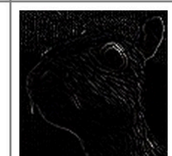
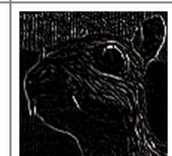
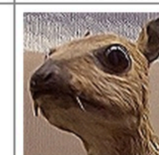
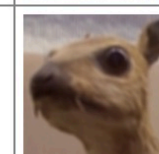
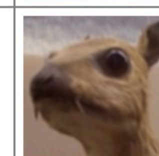
Figure 2. Gould performance showing note boundaries

Music Segmentation Analysis

- Basic idea (from image processing) uses a kernel or mask to modify data points according to their neighbors
- Each data point is replaced by the weighted sum of its neighbors * kernel values
- This is just convolution, but in 2 dimensions!…..

Music Segmentation Analysis



Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Music Segmentation Analysis

- A binary kernel finds boundaries along the axis where sections of music different from neighboring sections

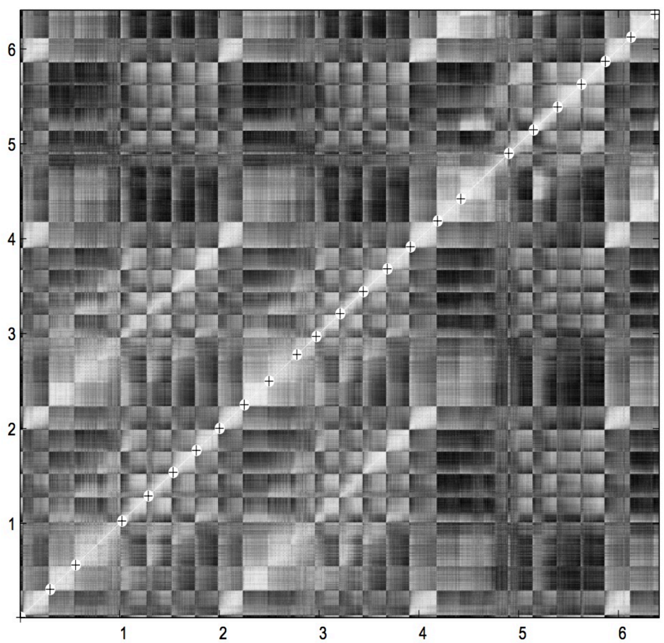


Figure 2. Gould performance showing note boundaries

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

Music Segmentation Analysis

- A Gaussian Kernel emphasizes changes at the center, and deemphasizes the edges (c.f. Hann Windows)

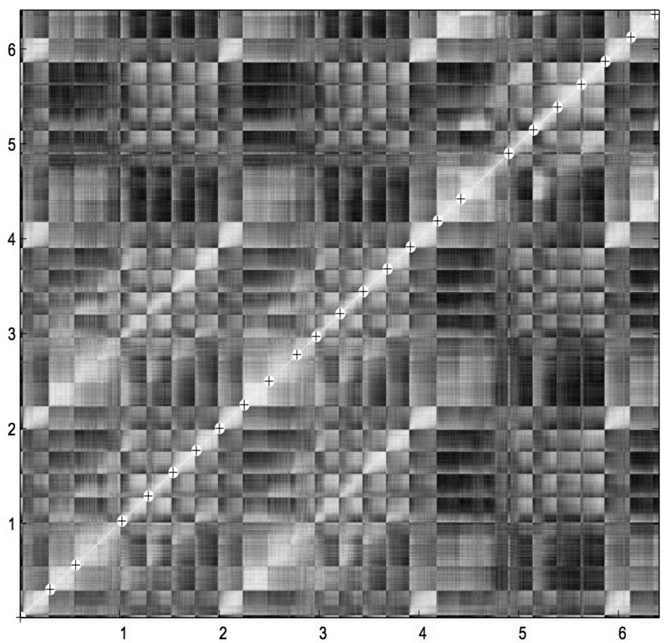


Figure 2. Gould performance showing note boundaries

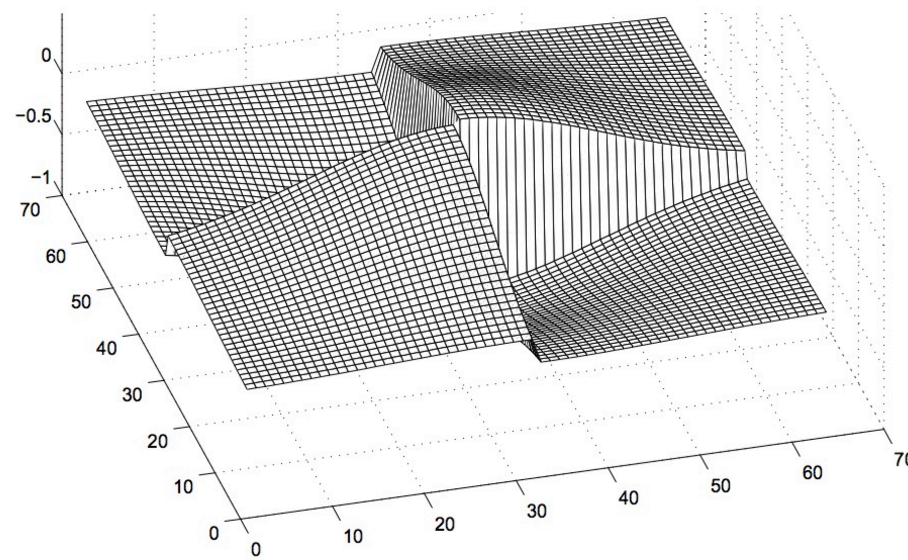


Figure 3. 64 x 64 checkerboard kernel with Gaussian taper

Music Segmentation Analysis

- The kernel is slid along the axis of the similarity matrix, and the value of the convolution is recorded for each time (in the center of the kernel):

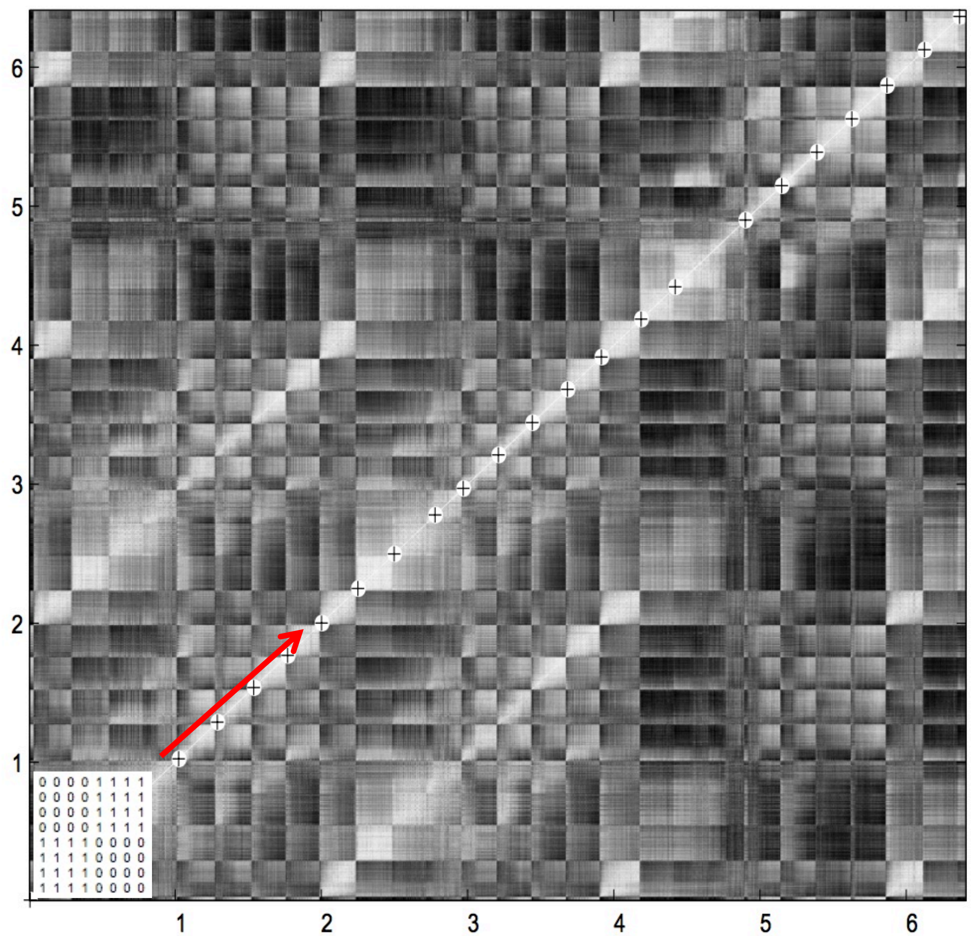


Figure 2. Gould performance showing note boundaries

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

Music Segmentation Analysis

- The kernel is slid along the axis of the similarity matrix, and the value of the convolution is recorded for each time (in the center of the kernel):

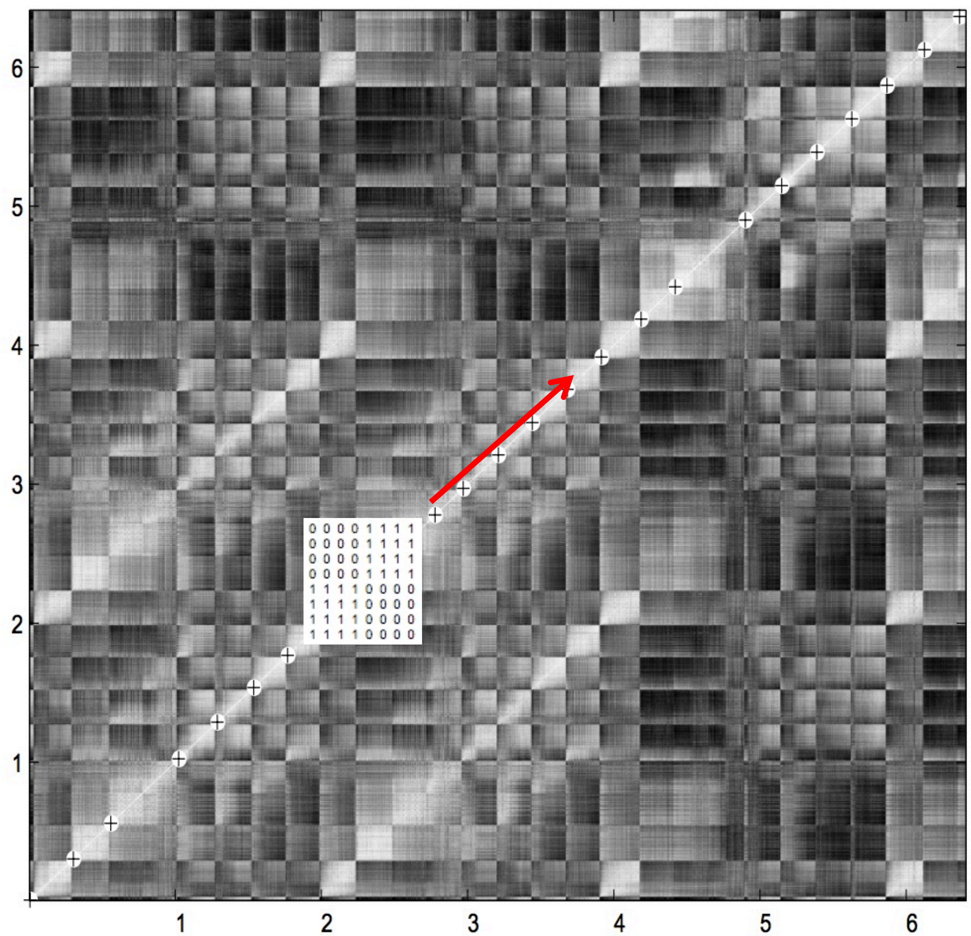


Figure 2. Gould performance showing note boundaries

$$
\begin{matrix}
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0
\end{matrix}
$$

Music Segmentation Analysis

- The kernel is slid along the axis of the similarity matrix, and the value of the convolution is recorded for each time (in the center of the kernel):

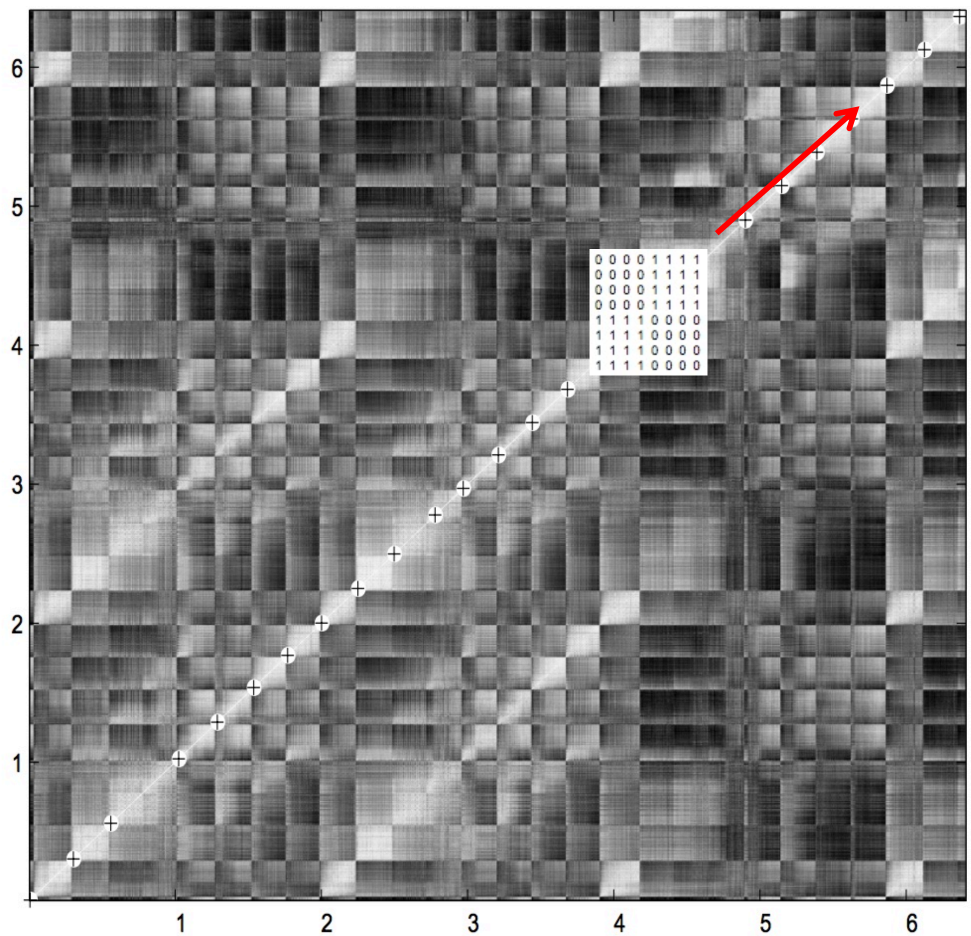


Figure 2. Gould performance showing note boundaries

$$\begin{matrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{matrix}$$

Music Segmentation Analysis

- As the kernel is slid along the axis, the values calculated give us a "novelty score" for how much the music is changing at that point
- Different kernel types and sizes give a different perspective on the scale of the changes, from individual notes to large sections….
- Peak picking gives us the times where there is a potential start of a new segment of music: