

Rewriting & Music

11th International School on Rewriting
Paris, MINES ParisTech, 1-6 July 2019

florent.jacquemard@inria.fr



- | | | |
|---------------------------|--|---|
| <u>part 0.</u> (today) | Examples in Musical Creation
at different Representation Levels | acoustic/physical domain
& notated/symbolic domain |
| <u>part 1.</u> (today) | Sequential Music Representations
Melodic Similarity , Computational Musicology
Weighted String Rewriting Systems & Edit Distances | notated/symbolic domain |
| <u>part 2.</u> (tomorrow) | Tree-structured Music Representations
Music Notation Processing, Transcription
Term Rewriting Systems & Weighted Tree Automata | notated/symbolic domain |

(click on a part to jump to its first slide)

Part I

Automated Music Analysis & Computational Musicology

Sequential Music Representations Melodic Similarity Measures

Weighted String Rewriting String Edit Distances

with

Algomus (Mathieu Giraud, Lille I)

Vertigo team (Philippe Rigaux, CNAM)

IReMus (Christophe Guillotel-Nothmann, CNRS)

Similarity in Music & Applications

Similarity in Bases of Audio Recordings

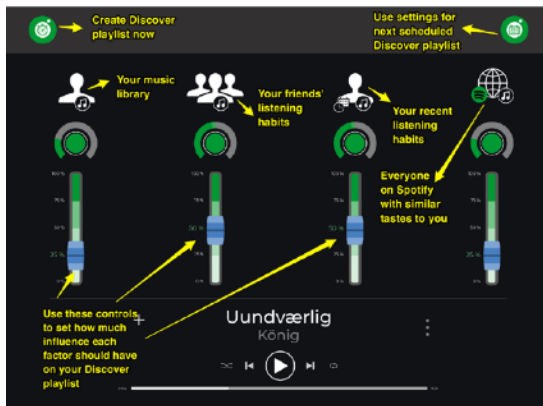
Audio Similarity

processing of audio signal
(low-level content features)

Music Information Retrieval

applications in audio databases (streaming platforms)

- automated classification (genre, mood, rhythm...)
- version identification (covers songs, opus retrieval)
- recommendation
- detection of plagiarism, apocrypha
- search and querying (query by humming)




ML Mixer Recommender System

Similarity in Music Score Databases

Symbolic Similarity

processing of symbolic music
representations
(high-level content description)

- Computational Musicology (corpora studies)
- Music Education, Composition
- Libraries

 <http://neuma.huma-num.fr>
database of digital music scores
in MusicXML and MEI formats
rare corpora (preservation).
search and analysis tools for musicologists.

applications in music score databases

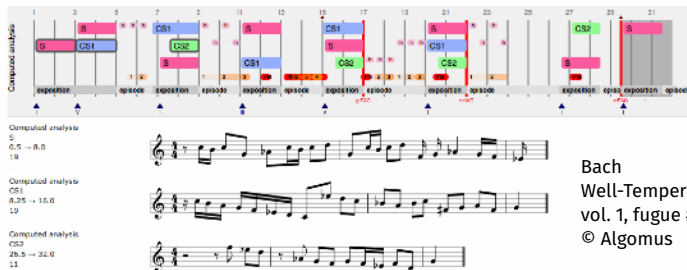
- automated classification
- version identification
- plagiarism detection
- search and retrieval
- automation of music analysis
- identification of structure

Vertigo team (CNAM, Cedric)
IReMus (Sorbonne U. CNRS)
BnF (French national library)



The screenshot displays the Neuma database interface. On the left, a list of search results is shown, including 'Greek, Orthodox Hymnals' and 'Greek, Orthodox Hymnals'. The right panel shows a detailed view of a selected score, featuring a musical staff with a treble clef and a piano keyboard visualization below it. The interface includes a search bar at the top and a list of results on the left.

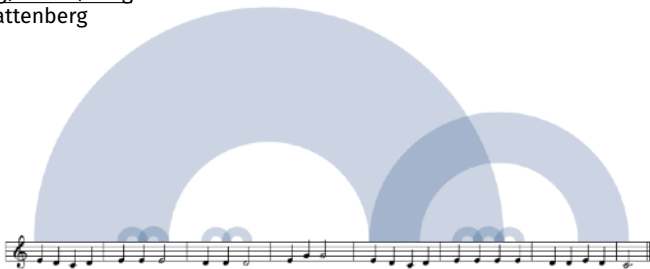
Computational analysis of written music: MIR on digital score corpora



automated formal analysis of music scores,
inference of high-level structure in scores, segmentation
identification of high-level descriptors (cadences, form...)
using melodic similarity measures to detect similar segments, repetitions...

Visualizing similarities (global structure)

turbulence.org/Works/song
Martin Wattenberg



folk song Clementine

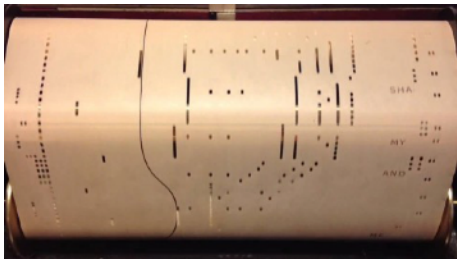


Goldberg variations

Piano-Roll Representation

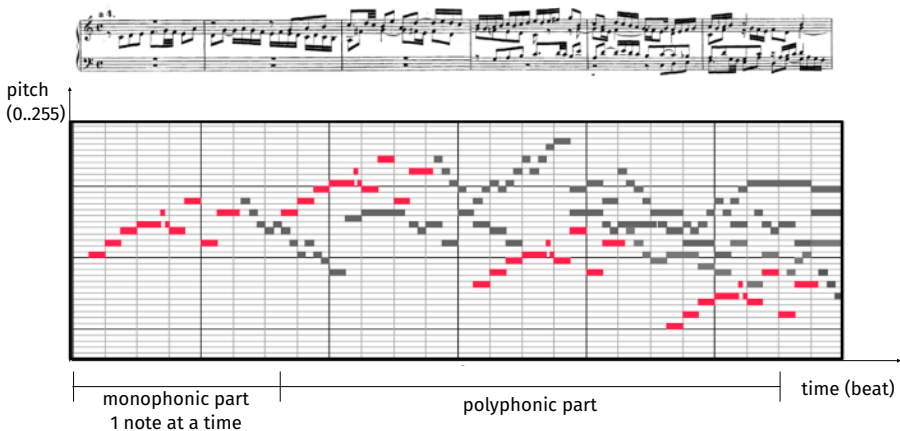


Colon Nancarrow



midi.org

Piano-Roll Representation



Symbolic Music Representations (monophony)

Representation by **1D strings** (monophonic melodies)
document and queries are sequences of symbols made of :

- pitch (or rest),
- duration (in nb of beats) or onset
(the end of a note is the start of the next)

► search for **exact match** :

with standard text searching algorithms: Knuth-Morris-Pratt, Boyer-Moore

Themefinder (regular expression match) for Essen Folksong Database

collection of European folk music available in quantized MIDI and kern**

► search for **approximate match** :

with similarity measures : edit distances

Musipedia (R. Typke)

main topic of lecture I

► **query by humming** and by tapping

MUSART, Musipedia (score bases)

SoundHound (audio base)

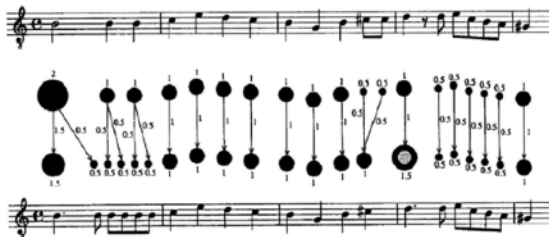
Symbolic Music Representations (polyphony)

Representations by **points in a 2D space** (polyphonic scores)
document and queries are finite sets of events made of:

- onset time,
- pitch,
- duration.

Geometric Methods

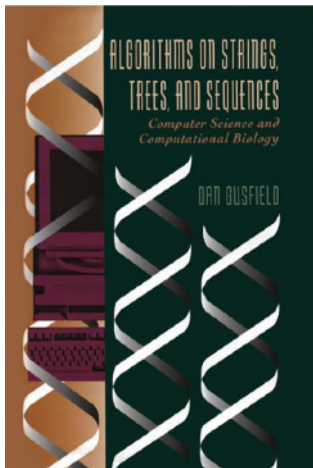
- ▶ **exact match**: query \subset document (modulo pitch shift of query)
- ▶ **approximate match**: document is superset of subset of the query
PROMS (M. Clausen, R. Engelbrecht, D. Meyer, and J. Schmitz)
- ▶ set comparison using transportation distances (EMD)
for comparing sets



Approximate String Matching & Edit-Distance Computation

measuring the similarity of sequences of symbols
applied to

- **text processing**
 - spell correction
 - plagiarism detection
 - file diff
(with variant: LCS)
- approximate search in documents
- Information Extraction
Named Entity Recognition and Entity Coreference
- **Molecular Biology** (DNA or protein sequences)
sequence di-similarities reflect biological events (mutations...)
sequence similarity implies functional or structural similarity
- Speech Recognition
(with variant: Dynamic Time Wrapping)



Dan Gusfield

Algorithms on strings, trees and sequences
Computer Science and Computer Biology
Cambridge University Press

Chapters on Approximate String Matching

- algorithms
- discussion on biological problems

In this presentation:

- insight of some algorithms in SRS settings
- discussion on musical relevance & extensions

String Similarity

What is the difference between:

1. cat and cats
2. cat and cut
3. sunday and saturday
4. intention and execution
5. vintner and writers

String Similarity

What is the difference between:

1. cat and cats 1 letter
2. cat and cut
3. sunday and saturday
4. intention and execution
5. vintner and writers

hint: align the strings by padding with _
in order to minimize non-matching slots.

1.

c	a	t	-
c	a	t	s

String Similarity

What is the difference between:

1. cat and cats 1 letter
2. cat and cut 1 letter
3. sunday and saturday
4. intention and execution
5. vintner and writers

hint: align the strings by padding with _
in order to minimize non-matching slots.

- | | | | | | | | | | |
|----|---|---|---|---|--|----|---|---|---|
| 1. | c | a | t | - | | 2. | c | a | t |
| | c | a | t | s | | | c | u | t |

String Similarity

What is the difference between:

1. cat and cats 1 letter
2. cat and cut 1 letter
3. sunday and saturday 3
4. intention and execution
5. vintner and writers

hint: align the strings by padding with _
in order to minimize non-matching slots.

1.

c	a	t	-
c	a	t	s
2.

c	a	t
c	u	t
3.

s	-	-	u	n	d	a	y
s	a	t	u	r	d	a	y

String Similarity

What is the difference between:

1. cat and cats 1 letter
2. cat and cut 1 letter
3. sunday and saturday 3
4. intention and execution 5
5. vintner and writers

hint: align the strings by padding with _
in order to minimize non-matching slots.

1.

c	a	t	-	
c	a	t	s	
2.

c	a	t
c	u	t
3.

s	-	-	u	n	d	a	y
s	a	t	u	r	d	a	y
4.

i	n	t	e	-	n	t	i	o	n
-	e	x	e	c	u	t	i	o	n

String Similarity

What is the difference between:

1. cat and cats 1 letter
2. cat and cut 1 letter
3. sunday and saturday 3
4. intention and execution 5
5. vintner and writers 5

hint: align the strings by padding with _
in order to minimize non-matching slots.

1. c a t _ 2. c a t
 c a t s c u t

3. s _ _ u n d a y
 s a t u r d a y

4. i n t e _ n t i o n
 _ e x e c u t i o n

5. v _ i n t n e r _
 w r i _ t _ e r s

Edition Rules

The SRS \mathcal{R}_0 over a finite alphabet Σ is defined as

$$\begin{aligned}\mathcal{R}_0 &= \{\varepsilon \rightarrow b \mid b \in \Sigma\} && \text{Insertion} \\ &\cup \{a \rightarrow \varepsilon \mid a \in \Sigma\} && \text{Deletion} \\ &\cup \{a \rightarrow b \mid a, b \in \Sigma, a \neq b\} && \text{Replacement}\end{aligned}$$

Rewriting step ($u, v \in \Sigma^*$):

$$\begin{aligned}u v &\xrightarrow{\mathcal{R}_0} u b v && \text{Insertion} \\ u a v &\xrightarrow{\mathcal{R}_0} u v && \text{Deletion} \\ u a v &\xrightarrow{\mathcal{R}_0} u b v && \text{Replacement}\end{aligned}$$

\mathcal{R}_0 is equivalent to the following TRS over $\Sigma \uplus \{\perp\}$

(symbols of Σ are unary, \perp constant):

$$\begin{aligned}\mathcal{R}_0 &= \{x \rightarrow b(x) \mid b \in \Sigma\} && \text{Insertion} \\ &\cup \{a(x) \rightarrow x \mid a \in \Sigma\} && \text{Deletion} \\ &\cup \{a(x) \rightarrow b(x) \mid a, b \in \Sigma, a \neq b\} && \text{Replacement}\end{aligned}$$

Rewriting problem:

reachability: given $s, t \in \Sigma^*$
does it hold that $s \xrightarrow[\mathcal{R}_0]{*} t$?

It is true for any s, t with \mathcal{R}_0 , because of rules Insertion and Deletion!

Quantitative Rewriting problem:

quantitative given $s, t \in \Sigma^*$
reachability: what is the minimal length of a rewrite sequence $s \xrightarrow[\mathcal{R}_0]{*} t$?

= how much 2 strings differ = edit-distance.

For $s, t \in \Sigma^*$, the Levenshtein distance $LD(s, t)$ is the minimal length of a rewrite sequence $s \xrightarrow{\mathcal{R}_0^*} t$.

Edit Distance problem:

given $s, t \in \Sigma^$*

compute $LD(s, t)$ along with a minimal $s \xrightarrow{\mathcal{R}_0^} t$.*

Example: $LD(\text{vintner}, \text{writers}) = 5$

$$\begin{array}{c} \text{vintner} \xrightarrow{\mathcal{R}_0^R} \text{wintner} \xrightarrow{\mathcal{R}_0^I} \text{wrntner} \xrightarrow{\mathcal{R}_0^D} \text{writner} \\ \xrightarrow{\mathcal{R}_0^D} \text{writer} \xrightarrow{\mathcal{R}_0^I} \text{writers} \end{array}$$

Is this sequence minimal?

Alignment

We denote by Σ_{-} the extension of a finite alphabet Σ with a new *space symbol* $- \notin \Sigma$.

Alignment

A pair $\langle s', t' \rangle$ of strings over Σ_{-} is an alignment of the pair $\langle s, t \rangle$ of strings over Σ if s', t' are obtained respectively from s and t by insertion of space symbols $-$ such that

- s' and t' have the same length,
- no position is labelled $-$ in both s' and t' .

Example: $s' =$ v _ i n t n e r _
 $t' =$ w r i _ t _ e r s

Observation 1.

Every alignment of $\langle s, t \rangle$ defines exactly one rewrite sequence $s \xrightarrow[\mathcal{R}_0]{*} t$.

Example:

$s =$	v	-	i	n	t	n	e	r	-
	↓R	↓I		↓D		↓D			↓I
$t =$	w	r	i	-	t	-	e	r	s

Converse?

Non-overlapping rewriting sequence (left-right strategy):

$$\begin{array}{c}
 \text{rewritable} \\
 \underbrace{} \\
 u \quad v_1 \ell v_2 \\
 \downarrow \\
 u \quad v_1 r \quad \underbrace{v_2}_{\text{rewritable}}
 \end{array}$$

where $\ell \rightarrow r$ is a string rewriting rule ($\ell, r \in \Sigma^*$), and $u, v_1, v_2 \in \Sigma^*$.

Observation 1'.

Every **non-overlapping** rewrite sequence $s \xrightarrow[\mathcal{R}_0]{*} t$ defines exactly one alignment of $\langle s, t \rangle$, and vice-versa.

- insert $_$ in s' at positions of Insertion,
- insert $_$ in t' at positions of Deletion.

The converse also holds: Consider the pairs of letters from left-to-right.

Observation 2.

Every rewrite sequence $s \xrightarrow[\mathcal{R}_0]{*} t$ of minimal length has no overlap.

Every 2 overlapping rewrite steps with \mathcal{R}_0 can be converted into strictly less rewrite steps (remember that \mathcal{R}_0 is "complete").

$uav \xrightarrow[\mathcal{R}_0]{R} ubv \xrightarrow[\mathcal{R}_0]{R} ucv$	$uav \xrightarrow[\mathcal{R}_0]{R} ucv \text{ if } a \neq c$
$uav \xrightarrow[\mathcal{R}_0]{R} ubv \xrightarrow[\mathcal{R}_0]{R} uav$	$uav \xrightarrow[\mathcal{R}_0]{0} uav$
$uv \xrightarrow[\mathcal{R}_0]{I} ubv \xrightarrow[\mathcal{R}_0]{R} ucv$	$uv \xrightarrow[\mathcal{R}_0]{I} ucv$
$uav \xrightarrow[\mathcal{R}_0]{R} ubv \xrightarrow[\mathcal{R}_0]{D} uv$	$uav \xrightarrow[\mathcal{R}_0]{D} uv$
$uv \xrightarrow[\mathcal{R}_0]{I} ubv \xrightarrow[\mathcal{R}_0]{D} uv$	$uv \xrightarrow[\mathcal{R}_0]{0} uv$

Hence, in order to compute $LD(s, t)$,
we can explore all alignments of $\langle s, t \rangle$ and find the best one.

But the number of alignments is **exponential** in the lengths of s and t .

Better solution: **think recursively**.

Let $s = a_1 \dots a_m$ and $t = b_1 \dots b_n$ in Σ^* .

For $1 \leq i \leq m$, $1 \leq j \leq n$, let $d_{i,j} = LD(a_1 \dots a_i, b_1 \dots b_j)$.

In particular, $d_{m,n} = LD(s, t)$.

$$d_{0,0} = 0$$

For $0 \leq i \leq m$, $0 \leq j \leq n$, let σ be the rewrite sequence $a_1 \dots a_i \xrightarrow{\mathcal{R}_0^*} b_1 \dots b_j$ of minimal length (its length is $d_{i,j}$).

By Observation 2, we can assume that σ has no overlaps, and commute its rewrite steps so that they are applied from **left to right**.

We consider the last (i.e. rightmost) rewrite step of σ .

There are 4 cases.

There are 4 cases for the last (*i.e.* rightmost) rewrite step of σ :

1. **Replacement** $a_i \xrightarrow{\mathcal{R}_0} b_j \ (a_i \neq b_j)$

Then $d_{i,j} =$

There are 4 cases for the last (*i.e.* rightmost) rewrite step of σ :

1. **Replacement** $a_i \xrightarrow{\mathcal{R}_0} b_j$ ($a_i \neq b_j$)

Then $d_{i,j} = d_{i-1,j-1} + 1$.

2. **Insertion** $\varepsilon \xrightarrow{\mathcal{R}_0} b_j$

There are 4 cases for the last (i.e. rightmost) rewrite step of σ :

1. **Replacement** $a_i \xrightarrow{\mathcal{R}_0} b_j \ (a_i \neq b_j)$

Then $d_{i,j} = d_{i-1,j-1} + 1$.

2. **Insertion** $\varepsilon \xrightarrow{\mathcal{R}_0} b_j$

Then $d_{i,j} = d_{i,j-1} + 1$.

3. **Deletion** $a_i \xrightarrow{\mathcal{R}_0} \varepsilon$

There are 4 cases for the last (i.e. rightmost) rewrite step of σ :

1. **Replacement** $a_i \xrightarrow{\mathcal{R}_0} b_j$ ($a_i \neq b_j$)

Then $d_{i,j} = d_{i-1,j-1} + 1$.

2. **Insertion** $\varepsilon \xrightarrow{\mathcal{R}_0} b_j$

Then $d_{i,j} = d_{i,j-1} + 1$.

3. **Deletion** $a_i \xrightarrow{\mathcal{R}_0} \varepsilon$

Then $d_{i,j} = d_{i-1,j} + 1$.

There are 4 cases for the last (i.e. rightmost) rewrite step of σ :

1. **Replacement** $a_i \xrightarrow{\mathcal{R}_0} b_j$ ($a_i \neq b_j$)

Then $d_{i,j} = d_{i-1,j-1} + 1$.

2. **Insertion** $\varepsilon \xrightarrow{\mathcal{R}_0} b_j$

Then $d_{i,j} = d_{i,j-1} + 1$.

3. **Deletion** $a_i \xrightarrow{\mathcal{R}_0} \varepsilon$

Then $d_{i,j} = d_{i-1,j} + 1$.

4. $a_i = b_j$.

Then $d_{i,j} = d_{i-1,j-1}$.

Summary:

$$d_{0,0} = 0$$
$$d_{i,j} = \min \left\{ \begin{array}{l|l} d_{i-1,j-1} + 1 & a_i \neq b_j \\ d_{i,j-1} + 1 & \\ d_{i-1,j} + 1 & \\ d_{i-1,j-1} & a_i = b_j \end{array} \right\}$$

We can implement the function d using these equation and call $d(m, n)$.

This top-down approach is very inefficient because of redundant recursive calls.

Better solution: **tabulation** (bottom-up approach):

fill a $m \times n$ matrix with the values of $d(i, j)$,
starting with the upper left corner $d(0, 0)$.

An optimal rewrite sequence can be computed simultaneously, or by
traceback.

Time complexity: $O(m.n)$

Space complexity: $O(m.n)$

Optimization:

when it is not required to compute an optimal rewrite sequence:

Time complexity: $O(LD(s, t). \min(m, n))$

Space complexity: $O(\min(LD(s, t), m, n))$

$$d_{0,0} = 0$$

$$d_{i,j} = \min \left\{ \begin{array}{l|l} d_{i-1,j-1} + 1 & a_i \neq b_j \\ d_{i,j-1} + 1 & \\ d_{i-1,j} + 1 & \\ d_{i-1,j-1} & a_i = b_j \end{array} \right\}$$

		w r i t e r s							
		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	5	6
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4	4	3	4	5	6
n	5	5	5	5	5	4	4	5	6
e	6	6	6	6	6	5	4	5	6
r	7	7	7	7	7	6	5	4	5

s' = v i n t n e r

traceback: we compute the matrix and in the same time add pointers according to the operation(s) selected in min: - right for insertion - down for deletion - diagonal for replace or match. following a path acc. to these directions from $(0,0)$ to (m, n) permit to reconstruct a rewrite sequence. in time $O(m + n)$.

In general, the Dynamic Programming techniques consist in:

- divide the problem into subproblems
defining a recurrence relation
- store the computed values in a table.

We can be more general:

every rule $\ell \rightarrow r$ of \mathcal{R}_0 is associated a non-zero **weight value** $\delta(\ell \rightarrow r)$.

(WSRS)

Quantitative Rewriting problem:

quantitative *given* $s, t \in \Sigma^*$

reachability: *what is the minimal weight of a rewrite sequence* $s \xrightarrow[\mathcal{R}_0]{*} t$?

For more generality, weights could be defined in a semiring.

Mehryar Mohri

Semiring frameworks and algorithms for shortest-distance problems

Journal of Automata, Languages and Combinatorics 7(3) 2002

Computation of $D(s, t)$: The equations become:

$$d_{0,0} = 0$$
$$d_{i,j} = \min \left\{ \begin{array}{ll} d_{i-1,j-1} + \delta(a_i \rightarrow b_j) & | \quad a_i \neq b_j \\ d_{i,j-1} + \delta(\varepsilon \rightarrow b_j) & \\ d_{i-1,j} + \delta(a_i \rightarrow \varepsilon) & \\ d_{i-1,j-1} & | \quad a_i = b_j \end{array} \right\}$$

It is correct provided that **triangle inequality** holds (to avoid overlaps):

$$\delta(\ell \rightarrow s) \leq \delta(\ell \rightarrow r) + \delta(r \rightarrow s)$$

Melodic Similarity

Levenshtein Edit Distance

(minimal number of rewrite rules used to transform a string into another)

is relevant for

- text processing (spell correction...), and
- biological sequence comparison.
- For music analysis?

(monophonic) melody = sequence of symbols (with pitch, duration)

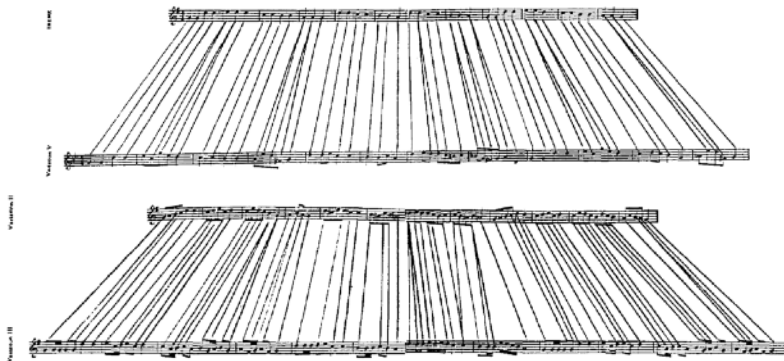
When computing melodic similarity as an edit-distance,
it is important to consider duration-preserving rewrite rules
→ problem with *insertion* and *deletion*.

rhythm is defined by regular patterns (**meter**)
meter should be preserved.

e.g. : what happens if one adds a random beat in a waltz?

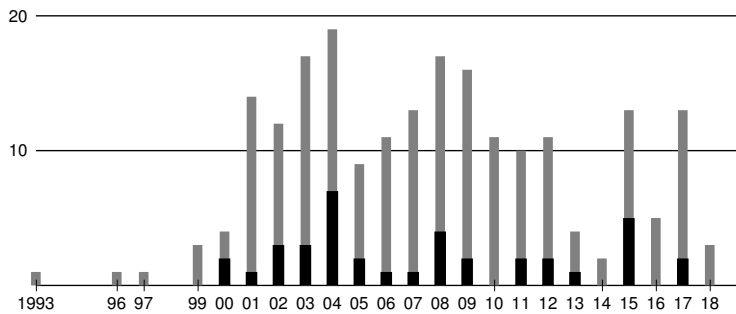
Marcel Mongeau and David Sankoff
Comparison of musical sequences
Computers and the Humanities, 24(3):161-175, 1990

inspired by the notion of **theme** and **variations** (= rewritings of theme)



Comparison of musical sequences

Marcel Mongeau and David Sankoff
Comparison of musical sequences
Computers and the Humanities, 24(3):161-175, 1990



Citations of the Mongeau-Sankoff algorithm throughout the years (data from semanticscholar)
Black bars represent papers presented at ISMIR (International Society for Music Information Retrieval Conference)

Esko Ukkonen
Algorithms for Approximate String Matching
Information and Control (64), 1985

Let \mathcal{R} be an arbitrary SRS over $\Sigma =$ finite set of string rewriting rules of the form $\ell \rightarrow r$ with $\ell, r \in \Sigma^*$, $\ell \neq r$
(called *editing operation set* in [Ukkonen 85]).

Every rule $\ell \rightarrow r \in \mathcal{R}$ is associated a weight value $\delta(\ell \rightarrow r) > 0$.

Edit Distance

For $s, t \in \Sigma^*$, $D_{\mathcal{R}}(s, t)$ is the minimal weight of a rewrite sequence $s \xrightarrow{\mathcal{R}}^* t$;
by convention, $D_{\mathcal{R}}(s, t) = +\infty$ if there is no such sequence.

Let $s = a_1 \dots a_m$ and $t = b_1 \dots b_n$ in Σ^* , and, for $1 \leq i \leq m$, $1 \leq j \leq n$

$$d_{0,0} = 0$$

$$d_{i,j} = \min \left\{ \begin{array}{l|l} d_{i-1,j-1} & a_i = b_j \\ d_{i-p,j-q} + \delta(\ell \rightarrow r) & \begin{array}{l} 0 \leq p \leq i \\ 0 \leq q \leq j \\ \ell = a_{i-p+1} \dots a_i \\ r = b_{j-q+1} \dots b_j \\ \ell \rightarrow r \in \mathcal{R} \end{array} \end{array} \right\}$$

Does $d(i, j)$ compute $D(a_1 \dots a_i, b_1 \dots b_j)$?

Restricted Edit Distance

For $s, t \in \Sigma^*$, $D'_{\mathcal{R}}(s, t)$ is the minimal weight of a *non-overlapping* rewrite sequence $s \xrightarrow{\mathcal{R}}^* t$;

by convention, $D'_{\mathcal{R}}(s, t) = +\infty$ if there is no such sequence.

Fact 1.

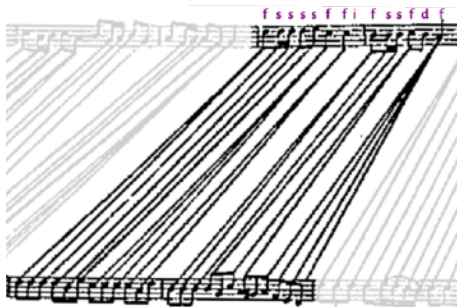
For all $s, t \in \Sigma^*$, $D_{\mathcal{R}}(s, t) \leq D'_{\mathcal{R}}(s, t)$.

It can be $<$ (in cases of overlap, examples later)

Fact 2.

For $s = a_1 \dots a_m$ and $t = b_1 \dots b_n$ in Σ^* , $1 \leq i \leq m$, $1 \leq j \leq n$,
 $d_{i,j} = D'_{\mathcal{R}}(a_1 \dots a_i, b_1 \dots b_j)$.

Example Variations K. 265



Alignments between variations 3 and 7 by M. Duchesnes on Mozart's *Ah vous dirai-je maman* K. 265
(figure from the original article of Montgeau & Sankoff 1990)

Variation 3 is a ternary meter and variation 7 in a binary one, making note-by-note alignment difficult, some steps rewrite one note into several.

They consider SRS with rules of the following forms:

$$\begin{array}{ccc} a & \rightarrow & b_1 \dots b_q \quad a, b_1, \dots, b_q \in \Sigma \quad \text{fragmentation} \\ \begin{array}{c} \text{musical staff with one note } a \end{array} & & \begin{array}{c} \text{musical staff with } q \text{ notes } b_1, \dots, b_q \end{array} \\ \\ a_1 \dots a_p & \rightarrow & b \quad a_1, \dots, a_p, b \in \Sigma \quad \text{consolidation} \\ \begin{array}{c} \text{musical staff with } p \text{ notes } a_1, \dots, a_p \end{array} & & \begin{array}{c} \text{musical staff with one note } b \end{array} \end{array}$$

deletion is a fragmentation with $q = 0$

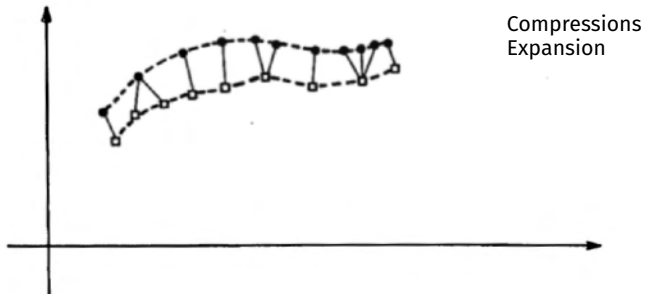
insertion is a consolidation with $p = 0$

replacement is a fragmentation with $q = 1$
(or consolidation with $p = 1$)

In this case the algorithm computing $D'_{\mathcal{R}}$ is:

$$d_{0,0} = 0$$

$$d_{i,j} = \min \left\{ \begin{array}{l|l} \begin{array}{l} d_{i-1,j-1} + \delta(a_i \rightarrow b_j) \\ d_{i-1,j} + \delta(a_i \rightarrow \varepsilon) \\ d_{i,j-1} + \delta(\varepsilon \rightarrow b_j) \end{array} & a_i \neq b_j \\ \begin{array}{l} d_{i-1,j-1} \\ d_{i-1,j-k} + \delta(a_i \rightarrow b_{j-k+1} \dots b_j) \\ d_{i-\ell,j-1} + \delta(a_{i-\ell+1} \dots a_i \rightarrow b_j) \end{array} & \begin{array}{l} a_i = b_j \\ 2 \leq k \leq j \\ 2 \leq \ell \leq i \end{array} \end{array} \right\}$$



Joseph B. Kruskal & Mark Libermann
The Symmetric Time-Warping Problem: from Continuous to Discrete
Time Warps, String Edits, and Macromolecules
The Theory and Practice of Sequence Comparison, CSLI Stanford 1999

elementary edit operations:

deletion, insertion, substitution +

- consolidation with cost = compress *substring* → *single char*
- fragmentation *id* = expanse *char* → *substring*

The image displays three staves of musical notation for Mozart's K625, illustrating edit operations between different versions of the music. The staves are labeled 'var. 1', 'theme', and 'var. 7'. The notation is in 2/4 time. Orange annotations highlight specific edit operations: 'c' for consolidation and 'f' for fragmentation. Arrows indicate the mapping between notes in different staves, showing how a single note in one version can correspond to a substring in another, or vice versa.

Mozart K625

Example Variations K.265 (2)

Alignments between variations 3 and 7 by M. Duchesnes
on Mozart's *Ah vous dirai-je maman* K. 265

Rewrite sequences with overlaps.

All considered consolidations and fragmentations preserve the total duration.

Diagram illustrating a musical score with three staves. The score is annotated with various symbols (C, S, F, f, d, i) and purple lines indicating the rewrite sequence. The annotations are as follows:

- Staff 1: C, S, F, C, d, f
- Staff 2: S, S, F, F, S, i, i, f
- Staff 3: F, i, F, i, i, f

This rewrite sequence involves
2 strict consolidation,
3 strict fragmentations,
3 other fragmentations,
6 s/d/i rules

total cost = $14 + 3.W_{dur.}$

Diagram illustrating a musical score with three staves. The score is annotated with various symbols (c, f) and purple lines indicating the rewrite sequence. The annotations are as follows:

- Staff 1: c, c, f, c, c, f
- Staff 2: f, f, f, f, f, f, f, f
- Staff 3: f, f, f, f, f, f

Here the rewrite sequence contains
only consolidations
and fragmentations
(11 rules, including 5 strict ones).

total cost = $11 + 6.W_{dur.}$

Mongeau-Sankoff Edit Distance

Edit dist. $D_{sd}(a, c) = 3$
 Best align. $D'_{sd}(a, c) = 3$
 Best align. $D'_{sd|CF}(a, c) = 3$



Edit dist. $D_{sd}(x, z) = 3$
 Best align. $D'_{sd}(x, z) = 3$
 Best align. $D'_{sd|CF}(x, z) = 3$



Edit dist. $D_{sd|CF}(a, c) = 2$



Edit dist. $D_{sd|CF}(x, z) = 2$



Align. $D'_{sd|cf}(a, c) = 1 + w_{pitch}$



Align. $D'_{sd|cf}(x, z) = 2 + w_{dur}$



Fact

In general it is undecidable whether $D_{\mathcal{R}}(s, t) < +\infty$ given $s, t \in \Sigma^*$ and \mathcal{R} over Σ with fragmentations and consolidations.

Encoding of the blank accepting problem for a Turing machine \mathcal{M} .

Every transition of \mathcal{M} is simulated by a combination of consolidation and fragmentation, or insertion, deletion.

Computable cases

$D_{\mathcal{R}}(s, t)$ can be computed in some specific cases:

1. when $\mathcal{R} = \mathcal{R}_0$
2. when \mathcal{R} contains only consolidations and deletions
3. when \mathcal{R} contains only fragmentations and insertions

1. $D_{\mathcal{R}_0} = D'_{\mathcal{R}_0}$ with triangle inequality.
2. PTIME construction of a weighted automaton $\mathcal{A}_s^{\mathcal{R}}$ such that $\mathcal{A}_s^{\mathcal{R}}(t) = D_{\mathcal{R}}(s, t)$.
3. inverse rules

Weighted Automata Construction: Example

$$\Sigma = \{i, t, u\}$$

all replacements,

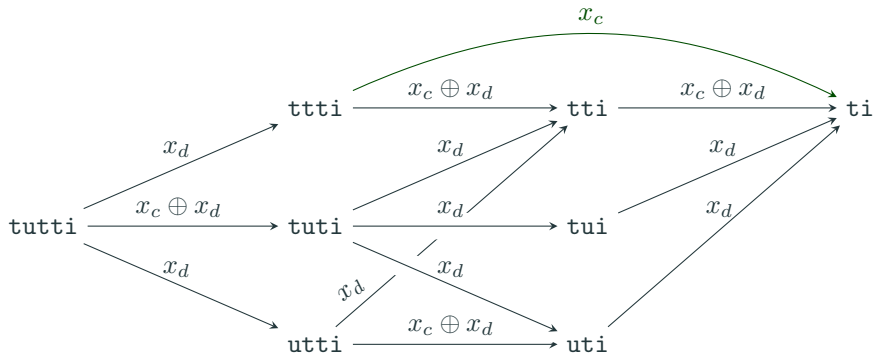
all insertions,

all deletions,

some consolidation and fragmentations.

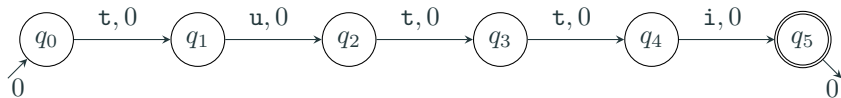
i	$\xrightarrow{x_s}$	t	ε	$\xrightarrow{x_i}$	i	i	$\xrightarrow{x_d}$	ε	ii	$\xrightarrow{x_c}$	i	i	$\xrightarrow{x_f}$	ii
i	$\xrightarrow{x_s}$	u	ε	$\xrightarrow{x_i}$	t	t	$\xrightarrow{x_d}$	ε	iii	$\xrightarrow{x_c}$	i	i	$\xrightarrow{x_f}$	iii
t	$\xrightarrow{x_s}$	i	ε	$\xrightarrow{x_i}$	u	u	$\xrightarrow{x_d}$	ε	tt	$\xrightarrow{x_c}$	t	t	$\xrightarrow{x_f}$	tt
t	$\xrightarrow{x_s}$	u							ttt	$\xrightarrow{x_c}$	t	t	$\xrightarrow{x_f}$	ttt
u	$\xrightarrow{x_s}$	i							uu	$\xrightarrow{x_c}$	u	u	$\xrightarrow{x_f}$	uu
u	$\xrightarrow{x_s}$	t							uuu	$\xrightarrow{x_c}$	u	u	$\xrightarrow{x_f}$	uuu

Weighted Automata Construction: Example



Rewrite sub-graph of tutti (only deletion and consolidation steps)

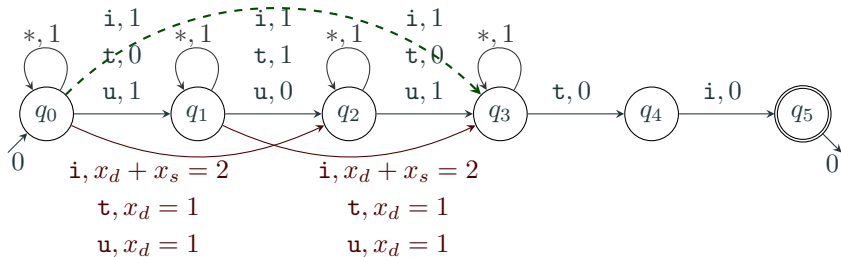
Weighted Automata Construction: Example



$$i, x_s + x_d + x_d = 3$$

$$t, x_d + x_c = 2$$

$$u, x_s + x_d = 2$$



$$x_i = x_d = x_c = 1$$

summary:

- sequential representation of monophonic melodies
computation of similarity
- Levenstein edit distance:
efficient computation but problem of relevance for melodic similarity
- Montgeau & Sankoff extension:
relevant musically (principle of theme - variation)
algorithm for computing alignments only
distance not computable
particular cases computable

diff tool for the comparison
of Music Score Files

Unix Diff

side-by-side comparison of 2 text files

- identify differences
- save in patch
- merge files

used in

- software development
- collaborative edition
- version control systems (git merge...)

```
comments.py@8c0de217c2 (read-only) x | comments.py x
-----
self.window.run_command('hide_panel', {'cancel': Tr
def show(self, comments = True, drafts = True):
    comments = self.prepare(comments, drafts)
+     if comments is None:
+         return
    items = []
-     if self.view_lines is not None and comments[0].col
+     if self.view_lines is not None and comments and com
        items.append(
            'caption': ['Add Draft Here'],
            'comment': comments[0],
            'on_over': lambda item: self.deactivate_com
            'on_select': lambda item: sublime.set_timeo
        )
    quick_panel(
        items + [self.create_comment_menu_item(comment,
            on_cancel=self.cancel
        )
-     if self.view_lines is not None and comments[0].collection.get_draft_at_line(comments[0].get_line()) is None:
+     if self.view_lines is not None and comments and comments[0].collection.get_draft_at_line(comments[0].get_line()) is None:

```

Changes: 39, Ignore CR/LF: Off, Ignore Whitespace: Off, Ignore Case: Off, Separate Missing Blocks: Off, Edit Mode Available, Line 665, Column 74

Sublimemerge 2 for macOS

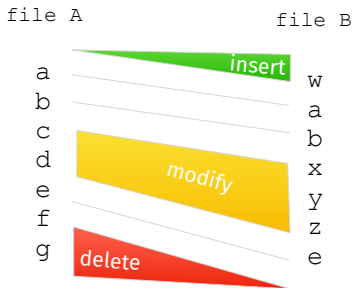
Longest Common Subsequence

used for diff of text files

informal objective:

given 2 text files (typically 2 versions of the same file)

identify the lines in common and the lines that differ



Let $L_{i,j}$ be the longest subsequence common to the first i lines of file $A = A_1, \dots, A_m$ and the first j lines of file $B = B_1, \dots, B_n$.

$$\forall i = 0, \dots, m \quad L_{i,0} = 0$$

$$\forall j = 0, \dots, n \quad L_{0,j} = 0$$

$$\forall i = 1, \dots, m, \forall j = 1, \dots, n \quad L_{i,j} = 1 + L_{i-1,j-1} \text{ if } A_i = B_j$$

$$L_{i,j} = \max(L_{i-1,j}, L_{i,j-1}) \text{ otherwise}$$

XML score files

for data exchange



```
<measure number="1">
  <attributes>
    <divisions>8</divisions>
    <key>
      <fifths>-1</fifths>
    </key>
    <beats>2</beats>
    <beat-type>4</beat-type>
    <staves>1</staves>
    <clef number="1">
      <sign>G</sign>
      <line>2</line>
      <clef-octave-change>0</clef-octave-change>
    </clef>
  </attributes>
  <note>
    <pitch>
      <step>C</step>
      <octave>5</octave>
    </pitch>
    <duration>6</duration>
    <voice>1</voice>
    <type>eighth</type>
    <dot/>
    <stem>down</stem>
    <beam number="1">begin</beam>
    <notations/>
  </note>
```

MusicXML (Finale)
(1 note)

```
<measure n="1" xml:id="m_sc_2" left="invis">
  <staff n="1">
    <layer n="1">
      <beam>
        <note xml:id="n_sc_6_0" pname="c" oct="5" dur="8" dots="1"/>
        <note xml:id="n_sc_7_0" pname="d" oct="5" dur="32"/>
        <note xml:id="n_sc_8_0" pname="c" oct="5" dur="32"/>
      </beam>
      <beam>
        <note xml:id="n_sc_9_0" pname="c" oct="5" dur="8"/>
        <note xml:id="n_sc_10_0" pname="c" oct="5" dur="8"/>
      </beam>
    </layer>
  </staff>
</measure>
<measure n="2" xml:id="m_sc_11">
  <staff n="1">
    <layer n="1">
      <beam>
        <note xml:id="n_sc_12_0" pname="c" oct="5" dur="16"/>
        <note xml:id="n_sc_13_0" pname="f" oct="5" dur="16"/>
      </beam>
      <note xml:id="n_sc_14_0" pname="f" oct="5" dur="4"/>
      <rest xml:id="n_sc_15_0" dur="16"/>
      <note xml:id="n_sc_16_0" pname="f" oct="5" dur="16"/>
    </layer>
  </staff>
</measure>
```

MEI (Verovio)
(2 bars)

XML formats for music score encoding are

- expressive
- verbose and **ambiguous**:

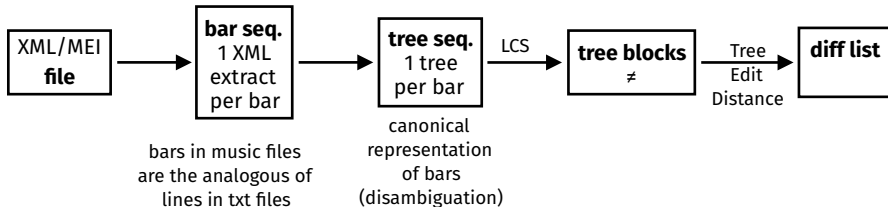
The same score can have many different XML encodings.

csq: it won't help to apply Unix diff directly to the XML (text) file

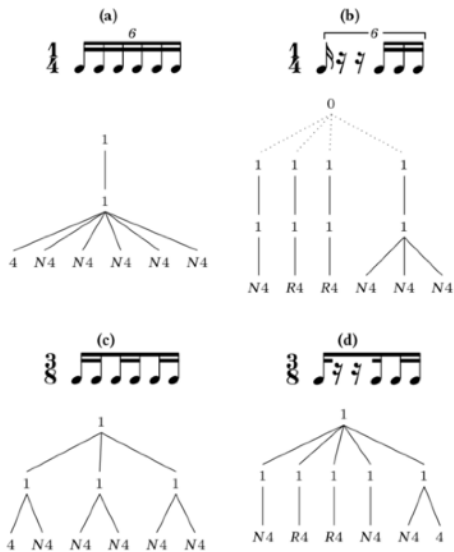
Christopher Antila, Jeffrey Treviño, Gabriel Weaver
A hierarchic diff algorithm for collaborative music document editing
TENOR 2017

Francesco Foscari (PhD)
A diff Procedure for XML/MEI Music Score Files

score file comparison proceeds in 2 steps
(after some pre-processing)



Tree Representation of XML score content



intermediate representation of the graphical content of a score with trees.

for **disambiguation**

It is a canonical representation:

2 different XML encodings of the same score elements will have the same tree representation.

edit primitives (on trees) \neq term rewrite rules

$$\begin{aligned}
 D(\varepsilon, s') &= \|s'\| & D(s, \varepsilon) &= \|s\| \\
 D(a(u).s, a'(u').s') &= \\
 \min \left\{ \begin{array}{ll} D(u.s, a'(u').s') + 1 & \text{(del-node)} \\ D(a(u).s, u'.s') + 1 & \text{(ins-node)} \\ D(s, s') + D(u, u') + \delta(a, a') & \text{(repl-node)} \end{array} \right.
 \end{aligned}$$

where $a(u)$, $a'(u')$ are trees, s , s' , u , u' are sequences of trees
 ε is the empty sequence, and

$$\delta(a, a) = 0$$

$$\delta(a, a') = 1 \text{ if } a \neq a' \text{ are inner symbols}$$

$$\delta(a, a') = \text{dist}(a, a') \text{ if } a, a' \text{ are constant symbols.}$$

Kaizhong Zhang and Dennis Shasha
 Simple fast algorithms for the editing distance between trees

Evaluation of XML-MEI music score files diff tool
on a dataset produced by **IReMus** (Sorbonne U. CNRS)
from a corpus of Bibliothèque nationale de France **BnF**, Gallica
containing 21 ouvertures of Jean-Philippe Rameau.

We diff one page OMReized from the manuscripts, and its manual correction.
Displayed differences can be useful for a fine detection of OMR errors.

(*) OMR = Optical Music Recognition = Music OCR



project Gioqoso between
Vertigo team (CNAM, Cedric)
IReMus (Sorbonne U. CNRS)
IRISA (Rennes)
BnF (French national library)

copyright

(**BnF** Gallica)

<https://gallica.bnf.fr>

OMRized version

Les surprises de l'amour

MEI exported with G2brando

Manual correction (ground truth)

Les surprises de l'amour

MEI exported with G2brando

diff marks:

- insertion
- deletion
- replacement (modification)

Summary of part I

- **sequential representations** of monophonic melodies for symbolic music analysis (musicology)
- **Levenshtein Edit Distance**
defined in SRS settings (rules insert, delete, replace)
quantitative reachability
efficient computation with Dynamic Programming
- **Extension by Mongeau & Sankoff**
Melodic Similarity (rules fragmentation, consolidation)
notion of overlaps
- diff utility for Score File
Longest Common Subsequence computation
Tree Edit Distance \neq Term Rewriting
- structured representations of music
music notation processing (next lecture)
- high-level (**musicological**) **features**
for audio Music Information Retrieval
(long term perspective in MIR community)

audio engineering community:
« *perceived musical information which,
though its existence is agreed by listeners,
stubbornly refuses to be extracted
from audio signals in isolation* »

(“glass ceiling” at about 70% accuracy
in the results achieved by audio processing alone)

Mozart, Symphony No. 40 in G minor, 2nd mvt., Andante

<https://youtu.be/14YKwZ5yYxw>
youtube channel **smalin**

