The SIMILE algorithms documentation 0.3

Daniel Müllensiefen & Klaus Frieler

October 9, 2006

Contents

1 Introduction

SIMILE is a software for the similarity comparison of two single line melodies. From the input of two symbolically encoded melody lines it computes one similarity value within the range from 0 (no similarity at all) to 1 (maximal similarity, identity) which represents the similarity relation according to one or more chosen musical dimensions. Formally, a similarity measure σ for the space of melodies $\mathcal{M}$ is a map

$$\sigma : \mathcal{M} \times \mathcal{M} \quad \rightarrow \quad [0,1]$$

with the following properties

1. $\sigma(\mu, \nu) = \sigma(\mu, \nu)$ (Symmetry)
2. $\sigma(\mu, \mu) = 1 \; \forall \mu \neq \emptyset$ und $\sigma(\emptyset, \mu) = 0$ (Self-identity)
3. Invariance under pitch transposition and tempo changes

This documentation describes on a rather abstract level the different algorithms implemented in SIMILE. For the usage of the programme please refer to SIMILE usage manual. SIMILE admits only melodies encoded in a special character separated data file format, named MCSV (Melody CSV). An in-depth specification of the MCSV format is given in the document "Melody-CSV File Format (MCSV)". Although the MCSV format is extensible to encode music with more than one voice sounding at a given time point and to polyphonic music, SIMILE accepts so far only MCSV files that encode only one series of musical notes, which is from now on called a "melody". A more detailed description of most of the algorithms employed in SIMILE including examples along with an application study can be found in Müllensiefen & Frieler (2004). The description of the algorithms employed in SIMILE is modular, this means for example that the main and basic transformations are described independently of the similarity measures. While a completely modular organisation of the SIMILE software package would be desirable for future development, SIMILE currently comprises only specific combinations of basic transformations, main transformations, general principles applied, and similarity algorithms. These specific combinations of algorithms are named by short abbreviations (max. 8 characters), which are listed in section 6.To get a first overview about how SIMILE works and transformations and algorithms are joined together, the interested reader is advised to have a brief look at section 6 at the end of this document.

2 Global algorithmic structure

There are several consecutive stages in which the data of two melodies are processed. The basic process flow is described in the Figure 1, which should be read from bottom to top:

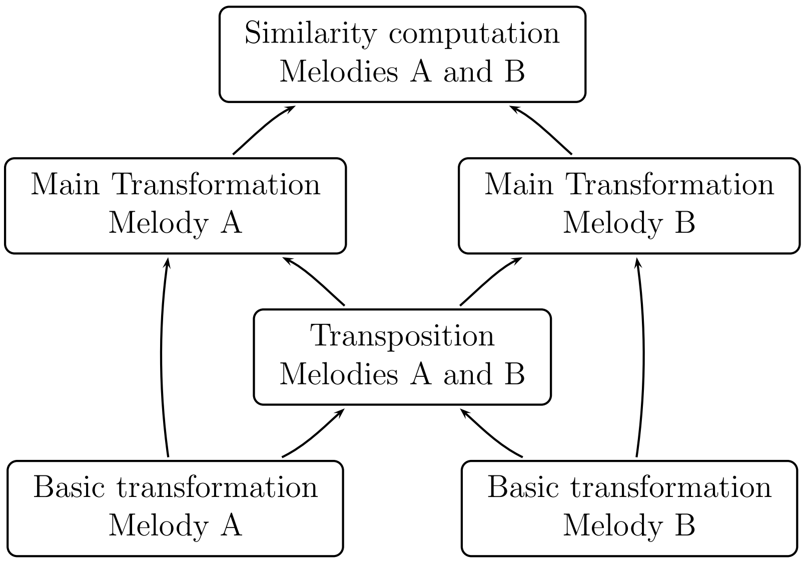

Figure 1: SIMILE's global algorithmic architecture

3 Data import and preprocessing

Generally the basic melodic events in SIMILE are sequences of onset/pitch pairs. These events are called notes, and in the remainder of the text we adopt the following notation for the more formal discussions.

$$n_i = (t_i, p_i)$$

3.1 Pitch information

The MCSV file provides pitch information p_i using MIDI pitch numbers (0-127), although a MIDI pitch of 0 is used to signal a rest and any event with pitch zero is discarded.

3.2 Time information

There is a bit of incoherence in the handling of time information in SIMILE, as there are 3 different sources for onsets and durations specified in the MCSV file format. We have onset time given in seconds or miliseconds [1] and onset time measured in MIDI-ticks. Furthermore, there is metric information provided for every note event, using bar number, beat location (one-based), and ticks (zero-based). For duration information we have a field for duration in seconds or miliseconds, a field for duration in ticks and a field for duration given in tatums. Onset in absolute time is currently only employed for the SimpleSegmenter

[1]Unfortunately no care was taken to specify which time unit (seconds or miliseconds) is actually used in a MCSV file, this sometimes needs to be guessed.

(cf. below). The durations actually used in SIMILE are inter-onset-intervals measured in tatums derived from metric information after a quantisation procedure (cf. below). The timebase (tatum) measured in ticks is calculated from the quantised metric information by taking the GCD of all IOI's. Note: Strictly speaking durations and inter-onset-intervals are different concepts, however often intermingled. We use currently only inter-onset-intervals in SIMILE. In further extension it is necessary to reflect this difference, because some algorithm (e.g. Termperleys Segmenter) relay on this information. Though one should note that, dependent on the data source, ofter no proper duration information is available, so inter-onset intervals need to be used instead.

4 The algorithms

4.1 Basic transformations

There are three basic transformations that manipulate the raw melodic data from the CSV input files that basically only contain onset times, durations and MIDI note numbers: pitch intervals, rhythmic quantisation, Interonsets intervals, and rhythmical weighting.

4.1.1 Pitch intervals

The use of a pitch interval representation is widely common in music analysis, because it ensures pitch transposition invariance, which is often desired. It is simple calculated by taking the differences between consecutive pitches (given as MIDI numbers):

$$\Delta p_n = p_{n+1} - p_n$$

.

4.1.2 Rhythmic quantisation and interonset intervals

An algorithm for rhythmic quantisation is used to assign onset values of note events to the nearest onset time point on a specified time grid. Duration values for the note events are calculated as well in a subsequent procedure. The quantisation algorithm works in the following steps:

- For the entire MCSV file it is checked whether onset times are on the quantisation grid, which means they must fall on a multiple of the value of the quantisation level. The default value of the quantisation level is 24, but the level can be adjusted using the -g option (see section 5.2 of the SIMILE User Manual).
- Because the given beat division is not always suited for quantisation of any level (e.g. Finale's MIDI export uses 1024 Ticks per beat, which is not suitable for triplets) the internal base tick division of the beat is changed to the lowest common multiple of original division and quantisation level.

- The onset values are then assigned to the nearest point on the metrical grid given by the quantisation. If two note events fall on the same time point the melody is considered invalid and SIMILE returns a warning message. Any similarity computation with at least one invalid file will result in similarity values of -1.
- A duration value for the time base is calculated by taking the minimum of the greatest common divisor of all durations as measured in ticks.We write ΔT for the tatum (time base) of a melody.

The calculation of IOIs is trivially done by taking delta time:

- The durations of note events are computed by taking the difference between two consecutive onset points:
$$\Delta t_i = t_{i+1} - t_i$$
- The musical durations (e.g. duration of semiquavers) for note events used in some of SIMILE'S algorithms are then calculated by subdividing the time base in ticks accordingly, formally
$$\Delta t_i^0 = \frac{\Delta t_i}{\Delta T} \in \mathbb{N}$$
thus, the information given in the durtic16 column of the MCSV file is ignored completely as a result.

4.1.3 Rhythmical weighting (RW)

Rhythmical weighting is way to assign weights to pitches in a pitch sequence that reflect the duration of a pitch in a melody. This is an useful transformation for algorithms that work on pitch information alone for still keeping some duration information . Every pitch is repeated as many times as there are smallest rhythmical units (tatums) in its duration. So, a sequence of a quaver of pitch c followed by a crotchet of a is represented in a weighted pitch sequence with a smallest rhythmical unit of a semi-quaver as: $c\ c\ a\ a\ a\ a$

Options for future implementations:

- Following ideas by Conklin (2006), further basic transformations to be derived are:
  - pitch class interval: melodic interval in modulo12 system
  - interval from a reference pitch, e.g. key of piece or tonic for that segment
  - interval direction: substitution of main transformation explained below
  - ratio of durations of consecutive notes
- The basic rules used for accent calculations (see 4.3.3) could be included in the Basic Transformations module

- An option to determine the segment of a melody that should only be processed would be helpful. Segment boundaries could be entered either by number of note from beginning or be bar, beat, and tick from beginning

---

4.2 Auxiliary principles

4.2.1 Transposition

As some main transformations and algorithms rather work on sequences of pitches than on sequences of intervals, a technique for transposing two melodies to the same pitch level is used before further transforming the pitch sequences. The transposition uses the Edit Distance similarity algorithm as explained below as a heuristic device and adds several constraints for plausible transpositions. The transposition algorithm is executed in the following steps:

1. A list of possible transposition candidates is determined from the according differences of the highest, the lowest, the most common pitches and the rounded mean pitches of the two melodies.

2. For each transposition hint the similarity between the two melodies is calculated according to:
$$\frac{\texttt{rawed} + \texttt{rawedw}}{2},$$
where `rawed` denotes the Edit Distance of the (raw, i.e. untransformed) pitch values of both melodies and `rawedw` is the rhythmically weighted Edit Distance of the raw pitch values.

3. The final transposition level is defined as the transposition step at which the greatest similarity value is found, and where the difference between the mean pitches of both melodies are not greater than 2.5 semitones.

Alternatives for determining best transposition: As the involved Edit Distance calculations are rather time consuming, the Krumhansl-Schmuckler algorithms could be utilized to the determine the overall key of melodies, and in an early pre-processing step all melodies could be transposed to a fixed tonality.

4.2.2 Horizontal shifting (HS)

Various similarity algorithms, especially those viewing melodies as vector representations, require melodies of equal note count. The strategy employed in SIMILE to solve this problem is to shift the shorter melody of two melodies of unequal length along the longer one event-wise while calculating a similarity value for each shift and taking the maxima of all values as the final similarity value. For a given melody μ we denote by μ_i^k the submelody of k notes that starts at the i-th note (adopting 1-based indexing here). Let μ, ν be two melodies with lengths $M > N$ and σ be a similarity measure, that requires equal length melodies. Then the shifted similarity value is given by

$$\sigma_{\text{shifted}}(\mu, \nu) = \max\{\max_{2 \le i \le i_0} \sigma(\nu_i^{N-i+1}, \mu_1^{N-i+1}), \max_{1 \le i \le M-N} \sigma(\mu_i^N, \nu)\}$$

In other words, we perform two shiftings. First, the shorter melody is sucessively shifted left starting at index 2 and ending at an index i_0 while cutting appropriate sections, calculating similarity values and taking the maximum. The offset i_0 is thereby determined as the index where $\nu_1^{i_0}$ represents 10% of the whole duration of the shorter melody, but never more than 8 tatums. When the beginning of both melodies coincide the shorter melody is subsequently shifted right until the endings of melodies coincide, perfoming the same cutting, calculating and maximum taking procedure. The overall maximum value of this process is the desired similarity value.

4.2.3 Squashing of value range (SVR)

Some similarity measures (e.g. correlation measures) can result in negative values. But in most applications similarity is conceptualised as a magnitude ranging from 0 (no similarity at all) to 1 (maximal similarity, identity), unless one is interested in retrograde similarity. To ensure that all computed values are between 0 and 1, squashing of the value range is used by setting all negative values to 0 and all values > 1 to 1.

4.2.4 Syncopicity

Within the accent transformation explained below some rules require information about the overall syncopation or syncopicity of a melody. For the calculation of the syncopicity of monophonic melodies meter information is needed that is generally provided in the MCSV input. Syncopicities can be calculated with regard to four different time levels: Half bar level, beat level, first subdivison, second subdivison. A metric level L_i can be conceived as a time grid $\{n\Delta T_i\}$, where time points of all lower level are contained in the grid of each higher level, specifically $\Delta T_i / \Delta T_{i-1} \in \{2, 3\}$ The syncopicity values are calculated as follows:

- Detemination of syncopated events. An event e is a syncope for a given level L_i iff
  1. it falls not on a grid point of this level, $e \notin L_i$
  2. it falls on a grid point of the next lower level $e \in L_{i-1}$
  3. it has an IOI that extends the time unit of the lower level, i.e. $IOI(e) > \Delta T_{i-1}$ or it is the last the note of a melody.
- Divide the number of syncopations by the number of notes, this is the syncopicity value for that level.

- The sum of the syncopicities of all levels is the overall syncopicity of the melody.

4.2.5 Phrase segmentation

For some main transformations and similarity algorithms a segmentation of a whole melody into melodic phrases is necessary. A segmentation structure can part of the input MCSV file. For example, segmentation models by Temperley (2001) and Cambouropoulos (2002) are part of the MELCONV file converting software, but can not be fed into SIMILE as segmentation information. However, there is a simple but in many cases convincing segmentation algorithm implemented in SIMILE itself called "SimpleSegmenter", which uses only information of large temporal gaps for segmentation marking. It is carried out in the following steps:

- The first note of a melody begins a phrase.
- Determine the mode of the note durations of a melody.
- Every note that terminates an interonset interval that is at least 3.9 times as long as the mode of the melody or is longer than 1.5 s is defined as the first note of a melodic phrase.
- All notes that fall between the first note n_{j_1} of phrase j and the first note n_{k_1} of phrase k (including n_{j_1}) belong to phrase j.
- 1-note phrases are allowed.

Options for future development: Integrate other segmentation models in SIMILE, e.g. Temperley (2001), Cambouropoulos (2002), Cambouropoulos-n-gram-parallelity (2003), Pearce (2006) + hybrid models made up of the aforementioned.

4.3 Main transformations

4.3.1 Transformations of pitch

4.3.1.1 Pitch ranking Elements like pitches or intervals can be meaningfully ranked. To this end, the numerical values of the N different elements in a melody are mapped onto the set of natural numbers from 0 to $N - 1$.

Options for future development: The ranking procedure should be made modular so that durations values and other representations could be ranked as well.

4.3.1.2 Contourisation

Contourisation is a method for deriving pitch values from a melody which represent rather the coarse directional motion than the actual sounding pitches. There are three variants of the contourisation algorithm implemented in SIMILE which only differ in their determination of changing notes. The contourisation algorithms can be described in the following steps:

- Determine all contour extremum notes. The contour extremum notes are the first note n_1, the last note n_N, and of every note n_i inbetween, where n_{i-1} and n_{i+1} are either both greater or both lower than n_i.
- As pure changing notes (notae cambiatae) don't make perceptually contour extrema, the changing notes are excluded from the set of potential contour extrema. This can be done in at least three ways which are implemented in SIMILE:
  - According to Steinbeck (1982), a changing note n_i is a potential contour extremum where the pitch values of $n_{i-2}, n_{i-1}, n_{i+1}$, and n_{i+2} are not all either lower or larger than the pitch of n_i. The changing notes are deleted from the set of contour extrema. This is called the Steinbeck contour.
  - According to Müllensiefen & Frieler (2004), a changing note n_i is a note where the pitches of n_{i-1} and n_{i+1} are equal. The changing notes are deleted from the set of contour extrema. This is called the M&F contour.
  - No changing notes are filtered from the set of contour extrema, all contour extremum notes are taken for contour calculation. This is called the natural contour.
- Calculate the gradient of the line between two subsequent contour extremum notes $n_i = (t_i, p_i)$ and $n_j = (t_j, p_j)$ $(j > i)$ by $m = \frac{p_j - p_i}{t_j - t_i}$
- Substitue the pitch values of every note $n_{i+k}, 1 < k < j - i$ between n_i and n_j by

$$p_{i+k} = p_i + m(t_{i+k} - t_i)$$

.

Options for future implementations:

- Use contour extrema only if they are accented notes or have a minimum accent value or if some other condition from the 34 accent rules is true $\Rightarrow$ Necessity to make all accent rules modular.
- 4th definition of changing notes to be implemented: changing note = auxiliary note: $p_{i-1} = p_{i+1} \wedge |p_i - p_{i+1}| < 3$ semitones.

4.3.1.3 Interval classes Pitch intervals are assigned to interval classes because it is assumed that human interval perception relies on only a limited number of intervalic movements. The assignment is done according to Table 1.

Table 1: Classification of intervals

Class	**Interval** (in semitones)	**Name**
-4	< -7	Large leap down
-3	-7, -6, -5	Leap down
-2	-4, -3	Large step down
-1	-2, -1	Step down
0	0	Same
1	1, 2	Step up
2	3, 4	Large step up
3	5, 6, 7	Leap up
4	>7	Large leap up

4.3.1.4 Interval direction Very much like interval classification, interval direction is a classification of intervals into a small number of categories. Three categories are used that capture only the directional movement from pitch p_i to pitch p_{i+1}. The categories are:

Class	**Condition**
Up	$p_{i+1} > p_i$
Same	$p_{i+1} = p_i$
Down	$p_{i+1} < p_i$

This transformation is sometimes called Parsons' code, and is also referred to as "contour" by some authors, a term which we reserve here for the linear interpolation between extrema (see 3.3.1.1).

4.3.1.5 Fourier transform According to Schmuckler (1999) the contour of a melody can be analysed by the Discrete Fourier Transform. The Fourier coefficients then are taken as a representation of the contour shape of a melody. A complex Discrete Fourier Transform is performed over the sequence of pitches or pitch ranks. The amplitudes of the real positive spectrum are kept as coefficients.

4.3.2 Transformations of rhythm

4.3.2.1 Duration classes As with pitch intervals, the durations of a note event f can be classified into a few classes (see table 2). The most frequent durational value (mode) of a melody is used as a reference duration estimating the beat interval.

Table 2: Classifications of durations

Class	**Duration of f** (in range of fractions of the mode of durations)	**Name**
0	f ≤ 0.45	very short
1	0.45<f≤0.9	Short
2	0.9<f ≤1.8	Normal (beat)
3	1.8<f ≤3.3	Long
4	f >3.3	Very long

4.3.2.2 Gaussification

Onsets of beats are gaussified by imposing a Gauss function with a fixed standard deviation and its mean over each onset. The results is a continous onset function for every time point of a melody. For a set of onsets t_n the rhythm gaussification is obtained by

$$g(t) = \frac{1}{N}\sum_{i=0}^{N-1} e^{-\frac{(t-t_i)^2}{2\sigma^2}}$$

The details of Gaussification are explained in Frieler (2004).

4.3.3 Accent transfomation

The idea behind accent transformation is that not all notes of a melody are perceptually of equal importance but that some notes have an accent, while others don't. So, accent transformation assigns a specific accent value to each note of a melody. The accent weight of a note is generated by mostly Gestalt-like rules that take the context of a note in a melody into account. The 34 rules are described in Pfleiderer, Müllensiefen & Frieler (2006). Each rule generates a binary output, i.e. it evaluates to 1 if it is true and to 0 if it is false. An optimal combination and weighting of a subset of rules is described in Pfleiderer & Müllensiefen (2006). The optimal rule combination for monophonic melodies is described by the following tree diagramme (figure 2), in which the numbers in the squares represent the accent weight to be predicted for a note event. The range of the accent values is $\{1,3\}$.

A different model of melodic accent calculates the accent weights of the individual notes of a melody as a weighted sum of the numerical values of a subset of rules. This model performed only slightly worse in a user study (Pfleiderer & Müllensiefen, 2006). The employed rules and their corresponding weights in this linear model are displayed in Table 3:

Options for future implementations:

- A scripting option to freely combine accent rules within a linear model (with weights) and as a tree model would be useful.

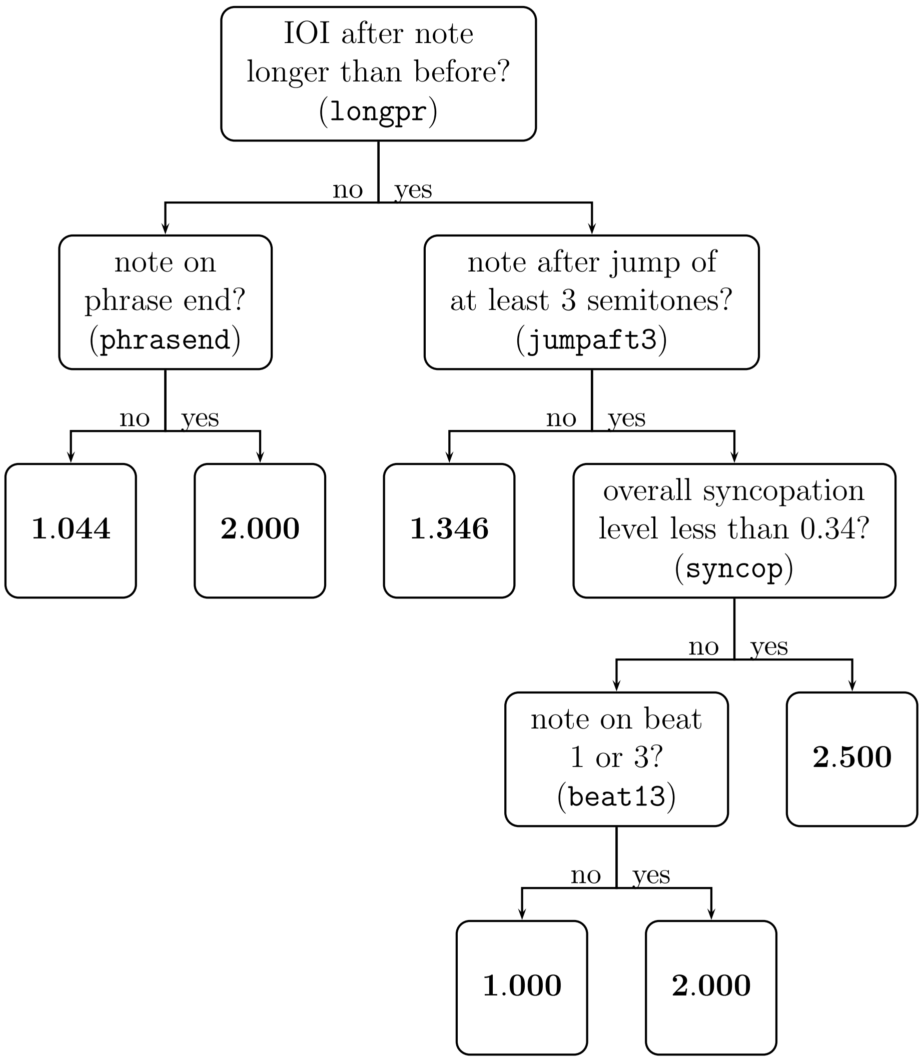
IOI after note longer than before? (longpr)
no
yes
note on phrase end? (phrasend)
note after jump of at least 3 semitones? (jumpaft3)
no
yes
no
yes
1.044
2.000
1.346
overall syncopation level less than 0.34? (syncop)
no
yes
note on beat 1 or 3? (beat13)
2.500
no
yes
1.000
2.000

Figure 2: Optimal rule combination for monophonic melodies

Table 3: Accent weights of linear accent model

Rule Name	**Weights**
constant	.973
PHRASEND	.413
BEAT13	.206
SHORTPR	-.275
LONGMOD	.287
PEXTREM	.361
JUMPAFT5	.463

- There are several ways to classify the resulting accent values, some of which have been implemented in earlier versions of SIMILE:
  - No classification, take accent values as they are
  - Weak Binary classification: 0 for all accent values below 1.5, 1 for all values above 1.5
  - Strong binary classification: 0 for all accent values not in the highest accent class (tree model) or below the 90% quantile of accent values (linear model), 1 for all other accent values.
  - Even Binary classification: 0 for all accent values in the three lower classes or below the median of accent values, 1 for all other accent values.
  - Ternary classification: 0 for all accent values in the lowest class or below 1.5, 2 for all accent values in the highest class (or above the 90% quantile), 1 for all other accent values.

Options for future implementations for main transformation in general: Incorporate Conklin's concepts of composite viewpoints and constructors (2006) and make them available for all types of basic transformations:

- Use value of note event only if the note event fulfills a certain condition, e.g. has melodic accent (Conklin: "select( )")
- Replace several identical consecutive values by one value (Conklin: "select([ ], new( )")
- List all occuring elements in element order (Conklin: "set( )")

All the features used for feature computation in the software package MELFEATURE could be used as main transformations, e.g. polynomial pitch contour, Huron pitch contour, range, central tendency, and variance of pitches and durations, complexity measures etc.

4.4 Similarity algorithms

4.4.1 Geometric distance-based measures

If a melody after transformation can be viewed embedded in a metrical vector space, the according metric of that space can always be used to construct various similarity measures, howver not in a unique way. From this theoretically large class of similarity measures in SIMILE currently only two measures based on the absolute norm of vectors of pitch intervals are used, called here difference measures.

4.4.1.1 Difference measures

The idea of the difference measures is to used the distance between two melodies μ_1, μ_2 of length N by calculating the average difference between the pitch intervals of two melodies. If the melodies to be compared are of unequal length horizontal shifting needs to be used. The mean absolute difference between two intervalls of two different melodies is calculated as:

$$\overline{\Delta p} = \sum_{i=1}^{N-1} |\Delta p_i^{(1)} - \Delta p_i^{(2)}|,$$

where $\Delta p_i^{(1,2)}$ are the pitch intervals of the two melodies. This expression can be viewed as the well-known absolute norm for the difference vector of pitch intervals.

$$\overline{\Delta p} = ||\vec{\Delta p}^{(1)} - \vec{\Delta p}^{(2)}||_1,$$

The two original difference measures are defined as follows:

4.4.1.2 `diffexp`

is defined as:

$$\sigma(\mu_1, \mu_2) = e^{-\frac{\overline{\Delta p}}{N-1}}$$

4.4.1.3 `diff`

is defined as:

$$\sigma(\mu_1, \mu_2) = 1 - \frac{\overline{\Delta p}}{(N-1)\Delta p_\infty}$$

with

$$\Delta p_\infty = \max_i |\Delta p_i^{(1)}| + |\Delta p_i^{(2)}|$$

4.4.2 Correlation measures

For correlating two melodies μ and ν, the elements (e.g. pitches) of both melodies can viewed as vectors in an appropriate vector space. To apply correlational measures both pitch vectors needs to have same dimensionality, If this is not the case horizontal shifting needs to be employed. Correlational measure are basically the Cosine of the angle between two vectors:

$$\cos\angle(u, v) = \frac{\langle u, v\rangle}{\sqrt{\langle u, u\rangle\langle v, v\rangle}}$$

As the Cosine gives always a number in the interval $[-1, 1]$, it suitable to apply squashing to all correlation measures.

In Simile two types of correlational measures are curently implemented, which differ in a translation of the vectors before taking the Cosine of the angle. For a n-dimensional vector u let $\bar{u}$ be a vector with the average

$$\bar{u} = \frac{1}{n}\sum_{i=1}^{n} u_i$$

as components. Likewise let u_0 be the vector with the first element of as components.

4.4.2.1 Pearson-Bravais correlation (PBC)

The Pearson-Bravais correlation coefficient σ_r is defined as the Cosine of the angle of the z-transformed vectors, i.e. each vector u is translated by a vector $\bar{u}$. The explicit formula is :

$$\sigma_r(u,v) = \frac{\sum_i (u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_i (u_i - \bar{u})^2 \cdot \sum_i (v_i - \bar{v})^2}}$$

4.4.2.2 Cross-Correlation (CC)

Another variant implemented in SIMILE are Cosine measures for pitch vectors translated by the vector u_0:

$$\sigma_{cc}(\mu_1, \nu_2) = \frac{\sum_i (u_i - u_0)(v_i - v_0)}{\sqrt{\sum_i (u_i - v_0)^2 \cdot \sum_i (v_i - v_0)^2}}$$

Options for future implementations: Take other points of reference, e.g. the most frequent, the highest or the lowest pitch.

4.4.2.3 Correlation of continuous functions

For cross correlation of gaussifications an appropriate inner product needs to be used as gaussifications are continuous functions.

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(t)g(t)dt$$

For gaussifications this inner product is always defined and can be integrated analytically:

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(t)g(t)dt \sim \sum_i \sum_j \exp - \frac{(t_i^g - t_j^f)^2}{2\sigma^2}.$$

Horizontal shifting is always employed in this case. Note, that this correlation measure is not tempo-invariant. One way to achieve this would be measuring onsets in units of the tatum of the according melody.

4.4.2.4 Fuzzy correlation (FC)

For a interval fuzzification we get for each element u_i of vector u subjected to this fuzzification a vector $\phi(u_i)$ of class memberships, which dimension is the number of fuzzy classes. To apply the cross correlation idea we can generalise the original formula of the inner product of two vectors to a inner product of vectors of vectors in the following way:

$$\langle\langle \phi(u), \phi(v) \rangle\rangle = \sum_k \langle \phi(u_k), \phi(v_k) \rangle = \sum_k \sum_i \phi_i(u_k)\phi_i(v_k),$$

and arrive at a cross-correlation

$$\sigma_\phi(u,v) = \frac{\langle\langle \phi(u), \phi(v) \rangle\rangle}{\sqrt{\langle\langle \phi(u), \phi(u) \rangle\rangle \langle\langle \phi(v), \phi(v) \rangle\rangle}}$$

Again, squashing can be applied, and if the dimensions of vectors u and v do not match, horizontal shifting should be used.

4.4.3 Symbolic measures

For symbolic measures melodies are regarded as strings of symbols. Two effective ways of computing the similarity, respectively distance between symbols strings used in SIMILE are Edit Distance (sometimes referred to as Levenshtein Distance) and various flavours of n-gram similarity.

4.4.3.1 Edit Distance (ED) The Edit Distance is the minimum number of operations it takes to transform one symbol string into another. The possible operations being insertion, deletion, and substitution. The actual calculation of the Edit Distance is carried out using dynamic programming and is not explained here. For a general reference regarding the algorithm, see e.g. Gusfield (1997). SIMILE only makes use of the so called global alignment which assumes that the two strings to be compared are of approximately equal length. In this case the maximal Edit Distance of two strings is equal to the length of the longer string. To convert the Edit Distance into a similarity measures with a range of values $[0, 1]$ we use the following standardisation:

$$\sigma(s,t) = 1 - \frac{d_e(s,t)}{\max(|s|,|t|)}$$

where $|s|$ and $|t|$ denote the element counts of strings s and t respectively, and $d_e(s,t)$ stands for the Edit Distance between strings s and t.

There are three variants of the Edit Distance used in SIMILE that differ in their cost functions for inserting/deleting and substituting elements and their scope of applicability:

- **Simple Edit Distance:** The costs of insertion/deletion and substituion of a symbol are always equal to 1.
- **Weighted Edit Distance:** This can be applied to strings of symbols that have a numerical value. The cost for inserting or deleting a symbol is exactly its numerical value. The cost for substituting a symbol with another symbol is the absolute difference between their numerical values
- **Phrase Edit Distance:** This variant is applied in phrase-base similarity measures. See Section 5 for details.

Options for future implementations: Implement the standard technique of local alignment for melodies of unequal length.

4.4.3.2 *n*-grams The core idea of the n-gram approach is to compare substrings of a certain length of the two strings s and t. The substrings are called n-grams, with n denoting the number of elements or the length of the substrings. Good discussions of using the n-gram approach for melodies general can be found in Downie (1999) and Uitdenbogerd (2002). Three different similarity measures are currently being implemented in SIMILE:

4.4.3.2.1 Sum Common Measure (SCM) We write s_n for the set of distinct n-grams in a string s having $|s_n|$ elements. The Sum Common Measure sums the frequencies of n-grams τ occurring in both strings:

$$c(s,t) = \sum_{\tau \in s_n \cap t_n} f_n(\tau) + f_t(\tau)$$

where $f_s(\tau)$ and $f_t(\tau)$ denote the frequencies of the n-gram τ in a string s and t respectively. The maximum frequency of an n-gram in a string s is $|s| - n + 1$, so the maximum value of the Sum Common Measure is $|s| + |t| - 2(n-1)$. A normalised similarity measure can then be constructed as:

$$\sigma(s,t) = \frac{c(s,t)}{|s| + |t| - 2(n-1)}$$

4.4.3.2.2 Count Distinct (Coordinate Matching) Measure (CM) The Count Distinct Measures bears some similarity with the Sum Common Measure with the only difference that the frequency of the n-grams common to both strings are not summed up, but the common n-grams are just counted. In order to arrive at a normalised similarity measure, the maximum count of distinct n-grams of either string is counted and taken and used as a denominator:

$$\sigma(s,t) = \frac{\sum_{\tau \in s_n \cap t_n} 1}{\max(|s_n|, |t_n|)}$$

4.4.3.2.3 Ukkonen Measure (UM) The Ukkonen Measure sums the differences between the frequencies of all n-grams not present in both strings. The formula is:

$$u(s,t) = \sum_{\tau \in s_n \cup t_n} |f_s(\tau) - f_t(\tau)|$$

As the Ukkonen Measure is a distance measure in its original definition, we normalise by the maximum possible number of n-grams and subtract the result from 1:

$$\sigma(s,t) = 1 - \frac{u(s,t)}{|s| + |t| - 2(n-1)}$$

4.5 Harmonic similarity measures

Though harmonic similarity measures could be described in the framework of Basic/Main Transformation/Similarity Algorithm they are somehow peculiar and we devote a single section to them. We start first by describing the original Krumhansl-Schmuckler algorithm and proceed by describing the actual in Simile implemented harmonic similarity measures. We do not make a strong distinction between implied tonality and implied harmony for several reasons. First, both concepts are strongly related. When implied harmony can be viewed as a gross description of contiguous sets of pitches, implied tonality can be viewed as a gross description of a contiguous set of implied harmonies. However, regarding monophonic melodies both concepts often coincide, respectively they could often only be separated by more or less arbitrary decisions. Secondly, one could argue, that the Krumhansl-Schmuckler algorithm actually measures implied harmonies as well as tonalities, because the used profiles have the strongest significance mainly on the values of the tonality representing triads. The last point being, that we deal here with comparisions of melodies and are not concerned with determining the "corrrect" implied harmonies or tonalities, so the distinction of harmony vs. tonality is for implicit comparisions probably not crucial. However, we prefer to speak of implied harmonies not tonalities, but sometimes this terms are used laxly.

4.5.1 The original Krumhansl-Schmuckler algorithm

The pitch values of a melody or a part of a melody can be condensed to a harmonic value, which reflects the tonality implied by the melodic notes in that particular segment. For the harmonic condensation the Krumhansl-Schmuckler algorithm is employed (Krumhansl, 1990), which can be summarised as follows:

1. For every segment sum the number of smallest durational units for all the IOIs of each pitch class. The result is a 12-dimensional vector where each element represents the time for which a pitch is sounding.
2. Correlate (using Pearson-Bravais-Correlation) this vector with the vector of the pitch weights for C-major and with the vector for c-minor as found by Krumhansl (see table 4 below).
3. Shuffle the two weight vectors around step by step and correlate at each shuffeling step the two mode vectors with the vector of the pitch durations.
4. The pitch class and mode that result in the highest correlation value with the duration vector are defined as the true implied tonality of the segment.
5. When calculating implied tonality bar-wise, the a weighted sum of the correlation values of bar $m-1$, m, and $m+1$ is calculated, where the values for m is weighted twice the value of $m-1$ and $m+1$.

Options for future development: Variants of the Krumhansl-Schmuckler-Algorithm can be implemented in the future. e.g. the variants proposed by Huron & Parncutt (1993) or Temperley (2001)

4.5.2 Harmonic Measures

There are several measures for harmonic similarity implemented in SIMILE, covering various ideas of how to determine implicit harmonies using variants of the Krumhansl-Schmuckler

Table 4: Krumhansl's tonality profile (Krumhansl, 1990)

Semitone	**Weights (Major)**	**Weights (Minor)**
0	6.35	6.33
1	2.23	2.68
2	3.48	3.52
3	2.33	5.38
4	4.38	2.60
5	4.09	3.53
6	2.52	2.54
7	5.19	4.75
8	2.39	3.98
9	3.66	2.69
10	2.29	3.34
11	2.88	3.17

tonality profiles and how to calculate a similarity from them. Most harmonic measures operate in a bar-wise manner, as bars are thought of being the main time-structures associated with implicit harmony. For a melodie μ we denote the pitch sequences of the i-th bar with $\beta_i(\mu)$, the number of bars for a melody is written $N_\beta(\mu)$. It is assumed that rhythmical weighting if it used is done first and preserves the bar structure. Furthermore, we denote Krumhansl's tonality profiles with H^M for major and H^m for minor. They can be viewed as 12-dimensional vectors with zero-based components $H^{\{m,M\}}(i)$, indicating the relative significance of each semitone step i class above a given tonic for this tonic. The values can be found in Table 4.

4.5.2.1 Mean harmonic correlation (MHC) Mean harmonic correlation for two melodies μ, ν is based on vector cross correlation. Let $N = \min(N_\beta(\mu), N_\beta(\nu))$ be the number of bars of the shorter melody. Correlation is done barwise and for each mode (major or minor) separately, where we take the average correlation of each bar. We define by

$$\bar{c}_M(\mu,\nu) = \frac{1}{N}\sum_{i=1}^{N} c_M(\beta_i(\mu), \beta_i(\nu))$$

and

$$\bar{c}_m(\mu,\nu) = \frac{1}{N}\sum_{i=1}^{N} c_m(\beta_i(\mu), \beta_i(\nu))$$

the mean correlations for each mode, then the overall harmonic similarity is given by

$$\sigma_h(\mu,\nu) = \max(c_M(\mu,\nu), c_m(\mu,\nu))$$

The single correlations for two bars β_1 and β_2 are given by

$$c_{\{m,M\}}(\beta_1, \beta_2) = \frac{\langle t^{\{m,M\}}(\beta_1), t^{\{m,M\}}(\beta_2)\rangle}{||t^{\{m,M\}}(\beta_1)||||t^{\{m,M\}}(\beta_2)||},$$

where we have introduced a 12-dimensional tonality vector function $t^{\{m,M\}}(\beta)$ with components

$$t_i^{\{m,M\}}(\beta) = \sum_{p\in\beta} H^{\{m,M\}}([p+i]_{12}), \tag{1}$$

with $[\cdot]_{12}$ denoting the residue class modulo 12. So for each pitch in each bar and each of the 12 possible tonalities the relative significance is summed, giving for each bar a tonality profile, where the components indicates the significance of the according tonality in this bar. This tonality profiles are compared with the standard Cosine similarity and the average overthe course of bars is calculated for each mode separately. The maximum thereof is the desired similarity value.

4.5.2.2 Harmonic cross correlation (HCC) This approach is similar to the foregoing measure, based on correlation, but using one tonaltiy vector for minor and major modes altogether. For two melodies μ_1, μ_2 let again $N = \min(N_\beta(\mu_1), N_\beta(\mu_2)$ be the number of bars of the shorter melody. We define now a 24-dimensional vector $t(\beta)$ for an arbitrary set of notes (bar) β by the intermingling the tonality vectors t^m, t^M from Eqn. 1 in the following way.

$$t_i(\beta) \quad = \quad \begin{cases} t_k^M(\beta) & \text{for } i = 2k \\ t_k^m(\beta) & \text{for } i = 2k+1 \end{cases}$$

We get now for each a melody a N-dimensional vector with these tonality vectors as components. We write for the components

$$c_n^{(i)} = t(\beta_n^{(i)}),$$

where the $\beta_n^{(i)}$ denote the n-th bar of melody μ_i. The idea is now to employ a cross-correlation for these vectors of vectors in the following way:

$$\sigma(\mu_1, \mu_2) = \frac{s^{12}}{\sqrt{s^{11}s^{22}}}$$

with the scalarproducts of vectors of vectors:

$$s^{(ij)} = \sum_{n=1}^{N} \langle c_n^{(i)}, c_n^{(j)}\rangle = \sum_{n=1}^{N}\sum_{k=0}^{23} t_k(\beta_n^{(i)}) t_k(\beta_n^{(j)})$$

As a Cosine similarity this measures can naturally subjected to squashing. There is a variant of this algorithm, where we don't use bars, but instead calculate one single 24-dimensional tonality vector for each melody. Thus we have:

$$\sigma'(\mu_1, \mu_2) = \frac{s'^{12}}{\sqrt{s'^{11}s'^{22}}}$$

with

$$s'^{(ij)} = \sum_{k=0}^{23} t_k(\mu_i) t_k(\mu_j)$$

4.5.2.3 Harmonic Edit Distance The basic idea of this algorithm is to represent a melody by a string of harmonic symbols. If this is achieved, the standard Edit Distance can be applied (and of course many more symbolic similarity algorithm, like n-grams.) To this end, we map each bar of a melody to the harmony/tonality which has the highest correlation according to the above defined 24-dimensional tonality vector, i.e. we have the implied harmony map

$$\beta_n(\mu) \quad \mapsto \quad \underset{0\leq i\leq 23}{\operatorname{argmax}}\, t_i(\beta_n(\mu)) \equiv T_n(\mu) \qquad (2)$$

As one might notice, this definition looks at the first glance different from the original Krumhansl-Schmuckler algorithm. However, for rhythmically weighted melodies this definitions are nearly equivalent, except for the bar-wise averaging step. To see this, we write down the Krumhansl-Schmuckler algorithm formally. First define joint shuffled tonality profiles by

$$H_i(k) \quad = \quad \begin{cases} H^M([k+i]_{12}) & \text{for } 0 \leq i \leq 11 \\ H^m([k+i]_{12}) & \text{for } 12 \leq i \leq 23 \end{cases}$$

For short, we will from now on write $[\cdot] \equiv [\cdot]_{12}$ for the residue class modulo 12. For each bar β we now define the original 12-dimensional duration-weighted pitch class vector $p(\beta)$ component-wise by

$$P_i(\beta) = \sum_{p\in\beta} \delta_{[i][p]} d(p),$$

with pitch class index $i \in [0, 11]$ and tatum duration $d(p) = \Delta t^0$ as defined above. The unaveraged implied harmony of the original algorithm is then given by pitch class indices giving the maximum correlation over shuffled tonality profiles and durational pitch class vector, i.e.

$$\beta \mapsto \underset{0\leq i\leq 23}{\operatorname{argmax}} \langle H_i, P(\beta)\rangle$$

The r.h.s evaluates now to

$$\begin{aligned} \langle H_i^\mu, P(\beta)\rangle &= \sum_{k=0}^{11} H_i^\mu(k) p_k(\beta) \\ &= \sum_{k=0}^{11} H([k+i]) \sum_{p\in\beta} \delta_{[i][p]} d(p) \\ &= \sum_{p\in\beta} H([p+i]) d(p) \end{aligned}$$

From this we see, that for rhythmically weighted melodies our definition, is identical to the original algorithm, except for the bar-wise averaging. For non-rhythmically weighted melodies we have a variant of the original with $d(p)$ set to one. Both variants are currently available in SIMILE.

4.5.2.4 Circle of fifths correlation (CCC) To exploit the reasonable distances of harmonies according to the circle of fifths, the circle cross correlation is defined in the following steps:

- Compute the implied harmonies $T_n = T(\beta_n(\mu))$ of each bar as described in equation 2, arriving at a value from 0 to 23 representing one of the 24 possible keys.
- Transform this value to a relative position on the unit circle by using an angular variable in steps of $\omega_0 = \frac{2\pi}{12}$. The circle of fifths can be conveniently represented by multiplying a pitch class index $[i]_{12}$ by 7 and taking again the pitch class, i.e. $[7i]_{12}$.

  $$\omega(T) \quad = \quad \begin{cases} \omega_0[7T]_{12} & \text{for } 0 \leq T \leq 11 \\ \omega_0([7T+3]_{12} + \alpha) & \text{for } 12 \leq T \leq 23 \end{cases}$$

  With a constant α, $0 < |\alpha| < 1$, which determines the twist of the circle of fifths for the minor modes to their relative major modes. In SIMILE α is currently fixed to $+\frac{1}{2}$.
- The similarity of two points on the circle of fifths is then defined as the cosine of the difference of their angles:

  $$r_k = \cos(\omega(T_k^1) - \omega(T_k^2))$$
- The mean of the similarities for all bars is computed by

  $$r = \frac{1}{N}\sum_{1}^{N} r_k$$
- Squashing is applied by setting all negative values to 0

Options for future development Correction: Shift minor keys by $-\frac{\omega}{2}$, because a-minor is closer to (has more notes in common with) F-major than to G-major. Alternative harmonic measures:

- Use a different squashing function, particularly $f(x) = \frac{1+x}{2}$ to get non-zero distance values for remote keys as well.
- Count notes in common as distance between the two keys
- Use harmonic distance measure from Woolhouse et al. (2006): $r = \sqrt{D} + 1$ with D being the distance in steps on the circle of fifths.

4.6 Hybrid similarity measures

In comparison with human similarity judgements (see Frieler & Müllensiefen, 2004), hybrid models consisting of two or more algorithmic chains of transformations and similarity algorithm have proven to have far more predictive power than only one algorithmic chain. This way pitch and duration information can be processed simultaneously. From the empirical data three hybrid meaures were found as weighted sums of the outputs of individual algorithmic chains which are named here according to table 6. The hybrid models differ in the context of a melody comparison. As the hybrid models result from optimisation with linear regression their range is not limited to the range of 0 to 1. To ensure that a proper similarity is received, squashing must be applied. SIMILE provides a general way to combine all hard-wired similarity measures to arbitrary weighted sums of two-way products of similarity measures. Please refer to the SIMILE user manual for more information.

4.6.1 `opti1`

For comparisons where there are only variants of the same melody to compare `opti1` is defined as follows:

$$\texttt{opti1} = 0.479 \cdot \texttt{rawedw} + 0.407 \cdot \texttt{ngrcoord}$$

`opti1` is sensitive to small changes in the pitch structure, and should be employed when the context of a specific melody is firmly established and the similarity of fine variants ought to be measured.

4.6.2 `opti2`

`opti2` is designed for situations where only the similarity of variants of one melody are of interest, but different melodies are also part of the context. The definition of `opti2` is:

$$\texttt{opti2} = 0.322 + 0.37 \cdot \texttt{rawedw} + 0.24 \cdot \texttt{ngrcoord}$$

`opti2` consists of the same algorithmic chains as opti1, but with a different relative weighting of the individual chains and it comprises a constant as well. Overall it produces generally higher similarity values than `opti2` .

4.6.3 `opti3`

`opti3` can generally be applied in situations, where similar melodies should be spotted within a context of similar and unsimilar melodies, e.g. in large melody collections. `opti3` is defined as:

$$\texttt{opti3} = 0.505 \cdot \texttt{ngrukkon} + 0.417 \cdot \texttt{rhythfuzz} + 0.24 \cdot \texttt{harmcore} - 0.146$$

`opti3` incorporates pitch and rhythmic information as well as harmonic content.

Options for future development: A general hybrid model scripting option might be useful. Algorithmic similarity chains could be combined in different types of prediction models, e.g. weighted linear models, trees, or rule based models.

5 Phrase-based similarity calculations

To calculate the similarity of two melodies μ and ν a phrase-based similarity calculation can be used as it is currently implemented in SIMILE. The idea behind this algorithm is that two melodies can be conceptualized as a sequence of phrases rather than a sequence of notes. The rationale behind this is the view that phrases are the true basic building blocks of melodies, roughly analogous to words in natural language as compared to syllables. To exploit this idea for a phrase-based similarity measure, a modified Edit Distance for sequences of phrases is used which is based on an arbitrary similarity measure. The algorithm comprises the following steps

1. Split the two melodies μ, ν into phrases using the SimpleSegmenter algorithm explained in section 4.2.5. This yields two "strings" μ_P, ν_P of phrases.
2. Apply a modified Edit Distance d_σ to the two strings of phrases with a cost of 1 for insertion and deletion of a phrase and a cost of $1 - \sigma(p_i, q_j)$ for substituting phrases p_i and q_j, where σ can be any similarity measure for phrases.
3. The global similarity of the two melodies is calculated as:

$$\sigma_P(\mu, \nu) = 1 - \frac{d_\sigma(\mu_P, \nu_P)}{\max(|\mu_P|, |\nu_P|)}$$

Options for future development: A method for phrase comparison that is not based on Edit Distance would be desirable. Similar phrases that are in distant places in the two melodies should be penalised less. A similarity measure that takes a matrix of phrase similarities between the two melodies into account might be possible.

6 Overview of algorithms and algorithmic chaims implemented in Simile

Table 6 gives a short definition of all similarity measures - or more precisely all algorithmic chains - implemented in SIMILE, listing their abbreviated name and the specific combination of the basic transformations, main transformations, and similarity algorithms employed for each measure. The following abbreviations are used: ED = (Simple) Edit Distance, PBC = Pearson-Bravais-Correlation, CC = Cross-Correlation, CCC = Circle of Fifths Cross Correlation, MHC = Mean Harmonic Correlation, HCC = Harmonic Cross-Correlation, FC = Fuzzy Correlation, SC = Sum Common Measure, CM = Coordinate Matching, UM = Ukkonen Measure, HS= horizontal shifting, RW = rhythmic weighting, SVR = Squashing.

Table 5: Algorithmic chains implemented in SIMILE

Name	**Basic Transformation**	**Main Transformation**	**Auxiliary Principle Applied**	**Similarity Algorithm**
`rawEd`	Pitch	-	-	ED
`rawEdw`	Pitch	-	RW	ED.
`rawPC`	Pitch	-	HS	PBC
`rawPCSt`	Pitch	-	HS, SVR	PBC
`rawPCw`	Pitch	-	HS, RW	PBC
`rawPCSt`	Pitch	-	HS, RW, SVR,	PBC
`rawCC`	Pitch	-	HS	CC
`rawCCw`	Pitch	-	HS, RW	CC
`conSEd`	Pitch	Contour Steinbeck	-	ED
`conSPC`	Pitch	Contour Steinbeck	HS,	PBC
`conSPCSt`	Pitch	Contour Steinbeck	HS, SVR	PBC
`conSCC`	Pitch	Contour Steinbeck	HS	CC
`conEd`	Pitch	Contour M&F	-	ED
`conPC`	Pitch	Contour M& F	HS	PBC
`conPCSt`	Pitch	Contour M&F	HS, SVR	PBC
`conCC`	Pitch	Contour M&F	HS	CC
`connEd`	Pitch	Contour natural	-	ED
`connPC`	Pitch	Contour natural	HS	PBC
`connPCS`	Pitch	Contour natural	HS, SVR	PBC
`connCC`	Pitch	Contour natural	HS	CC
`fourR`	Pitch	Ranking, Fourier Trans.	HS	PBC

Table 5: Algorithmic chains implemented in SIMILE

Name	**Basic Trans-formation**	**Main Trans-formation**	**Auxiliary Principle Applied**	**Similarity Algorithm**
`fourRSt`	Pitch	Ranking, Fourier Trans.	SVR	PBC
`fourRw`	Pitch	Ranking, Fourier Trans.	RW	PBC
`fourI`	Intervals	Fourier Trans.	-	PBC
`diffEd`	Intervals	-	-	ED
`diff`	Intervals	-	HS	Diff
`diffExp`	Intervals	-	HS	DiffExp
`diffFuz`	Intervals	Interval Classes	-	FC
`diffFuzC`	Pitch	Contour M&F, Interval Classes	-	FC
`bGrSumCo`	Intervals	-	-	2-gram: SC
`bGrUkkon`	Intervals	-	-	2-gram: UM
`bGrCoord`	Intervals	-	-	2-gram: CM
`bGrSumCR`	Intervals	Parsons Code	-	2-gram: SC
`bGrUkkonR`	Intervals	Parsons Code	-	2-gram: UM
`bGrCoorR`	Intervals	Parsons Code	-	2-gram: CM
`bGrSumCF`	Intervals	Interval Classes	-	2-gram: SC
`bGrUkkoF`	Intervals	Interval Classes	-	2-gram: UM
`bGrCoorF`	Intervals	Interval Classes	-	2-gram: CM
`bGrSumFR`	Duration	Duration Classes	-	2-gram: SC
`bGrUkkoFR`	Duration	Duration Classes	-	2-gram: UM
`bGrCooFR`	Duration	Duration Classes	-	2-gram: CM
`nGrSumCo`	Intervals	-	-	3-gram: SC
`nGrUkkon`	Intervals	-	-	3-gram: UM
`nGrCoord`	Intervals	-	-	3-gram: CM
`nGrSumCR`	Intervals	Parsons Code	-	3-gram: SC
`nGrUkkonR`	Intervals	Parsons Code	-	3-gram: UM

Table 5: Algorithmic chains implemented in SIMILE

Name	**Basic Trans-formation**	**Main Trans-formation**	**Auxiliary Principle Applied**	**Similarity Algorithm**
`nGrCoorR`	Intervals	Parsons Code	-	3-gram: CM
`nGrSumCF`	Intervals	Interval Classes	-	3-gram: SC
`nGrUkkoF`	Intervals	Interval Classes	-	3-gram: UM
`nGrCoorF`	Intervals	Interval Classes	-	3-gram: CM
`nGrSumFR`	Duration	Duration Classes	-	3-gram: SC
`nGrUkkoFR`	Duration	Duration Classes	-	3-gram: UM
`nGrCooFR`	Duration	Duration Classes	-	3-gram: CM
`qGrSumCo`	Intervals	-	-	4-gram: SC
`qGrUkkon`	Intervals	-	-	4-gram: UM
`qGrCoord`	Intervals	-	-	4-gram: CM
`qGrSumCR`	Intervals	Parsons Code	-	4-gram: SC
`qGrUkkonR`	Intervals	Parsons Code	-	4-gram: UM
`qGrCoorR`	Intervals	Parsons Code	-	4-gram: CM
`qGrSumCF`	Intervals	Interval Classes	-	4-gram: SC
`qGrUkkoF`	Intervals	Interval Classes	-	4-gram: UM
`qGrCoorF`	Intervals	Interval Classes	-	4-gram: CM
`qGrSumFR`	Duration	Duration Classes	-	4-gram: SC
`qGrUkkoFR`	Duration	Duration Classes	-	4-gram: UM
`qGrCooFR`	Duration	Duration Classes	-	4-gram: CM
`rhytGauss`	Rhythm	Gaussification	-	CC.
`rhytFuzz`	Durations	Duration Classes	-	ED
`harmCorr`	Pitch	Impl. Harm. Bar	RW	MHC
`harmCorK`	Pitch	Impl. Harm. Bar	-	DHC
`harmCorE`	Pitch	Impl. Harm. Bar	-	ED
`harmCorC`	Pitch	Impl. Harm. Bar	-	CCC

Table 5: Algorithmic chains implemented in Simile

Name	**Basic Transformation**	**Main Transformation**	**Auxiliary Principle Applied**	**Similarity Algorithm**
`harmCoKA`	Pitch	Impl. Harm. Total	-	HCC
`harmcoEA`	Pitch	Impl. Harm. Bar	-	ED
`harmcoCA`	Pitch	Impl. Harm. Total	-	CCC
`harcorrW`	Pitch	Impl. Harm. Bar	RW	MHC
`harcorKW`	Pitch	Impl. Harm. Bar	RW	HCC
`harcorEW`	Pitch	Impl. Harm. Bar	RW	ED
`harcorCW`	Pitch	Impl. Harm. Bar	RW	CCC
`harcoKAW`	Pitch	Impl. Harm. Total	RW	HCC
`harcoEAW`	Pitch	Impl. Harm. Total	RW	ED
`harcoCAW`	Pitch	Impl. Harm. Total	RW	CCC
`accents-opti`	1All	Accents Lin.	-	ED
`accents-opti`	2All	Accents Tree	-	ED

References

[1] Cambouropoulos, E. (1998). *Towards a General Computational Theory of Musical Structure.* PhD Thesis, University of Edinburgh

[2] Downie, J.S. (1999). "Evaluating a simple approach to music information retrieval." *Conceiving melodic n-grams as text.* PhD Thesis, University of Western Ontario.

[3] Frieler, K. (2004). "Beat and meter extraction using gaussified onsets." *ISMIR Proceedings Barcelona, 2004*

[4] Jones, M.R. (1987). "Dynamic pattern structure in music: Recent theory and research." *Perception & Psychophysics*, 41, p. 621-634.

[5] Krumhansl, C. L. (1990). *Cognitive foundations of musical pitch.* New York: Oxford University Press.

[6] Mongeau, M., and Sankoff, D. (1990). "Comparison of musical sequences". *Computers and the Humanities* 24, 161-175.

[7] Müllensiefen, D., and Frieler, K. (2004). "Cognitive Adequacy in the Measurement of Melodic Similarity: Algorithmic vs. Human Judgements". *Computing in Musicology* 13, 147-176.

[8] O'Maidin, D. (1998). "A Geometrical Algorithm for Melodic Difference in Melodic Similarity." Melodic Similarity: Concepts, Procedures, and Applications. *Computing in Musicology* 11. Ed. Walter B. Hewlett & Eleanor Selfridge-Field. Cambridge: MIT Press.

[9] Pfleiderer, M., and Müllensiefen, D. (2006). "The perception of accents in pop music melodies. *Proceedings of the 9th International Conference on Music Perception and Cognition, Bologna, 2006.* http://www.escom-icmpc-2006.org/pdfs/513.pdf.

[10] Schmuckler, Mark A. (1999) "Testing Models of Melodic Contour Similarity." *Music Perception*, Vol. 16, No. 3, 109-150.

[11] Steinbeck, W. (1982). *Struktur und Ähnlichkeit. Methoden automatisierter Melodieanalyse.* Kassel: Bärenreiter.

[12] Temperley, D. (2001). *The Cognition of Basic Musical Structures* Cambridge, MA: MIT Press.

[13] Uitdenbogerd, A. L (2002). *Music Information Retrieval Technology.* PhD thesis. RMIT University Melbourne Victoria, Australia