EFFICIENT MELODY RETRIEVAL WITH MOTIF CONTOUR CLASSES

Tillman Weyde
City University
School of Informatics
Department of Computing
London, UK
t.e.weyde@city.ac.uk

Christian Datzko
University of Osnabrück
Research Department of
Music and Media Technology
Osnabrück, Germany
christian@datzko.ch

ABSTRACT

This paper describes the use of motif contour classes for efficient retrieval of melodies from music collections. Instead of extracting incipits or themes, complete monophonic pieces are indexed for their motifs, using classes of motif contours. Similarity relations between these classes can be used for a very efficient search. This can serve as a first level search, which can be refined by using more computationally intensive comparisons on its results. The model introduced has been implemented and tested using the MUSITECH framework. We present empirical and analytical results on the retrieval quality, the complexity, and quality/efficiency trade-off.

Keywords: melody retrieval, motivic analysis, melodic similarity, retrieval efficiency

1 INTRODUCTION

Although audio retrieval has been in the focus of attention lately, melody retrieval based on symbolic music representations is important for databases like Themefinder[1] and Digital Tradition[2] and will likely become more important in the near future with the development of the MPEG standard on Symbolic Music Representation[3]. The most common approach for melody retrieval today is to compare a query to a melody that has somehow been extracted from a piece of music, comparing query and melody as a whole using an edit distance approach.

The basic idea of this paper is to use musical motifs for melody retrieval, similarly as words are used in text retrieval. Musical motifs, as defined by Riemann (1903), are the smallest musically meaningful parts of a melody. Retrieval based on motifs is independent of the size of the pieces in the database. Provided the voice structure is known, complete symphonies can be indexed and retrieved this way. This method captures similarities that result from recombination of motivic material. Similarities between motifs can also be taken into account.

Indexing techniques relying on manual extraction of themes etc. are costly and limit the search, therefore automatically indexing a complete piece of music is a way of efficiently making more information accessible. Work in this direction has been done by Clausen et al. (2001), whose system builds an index for every note in a piece – leading to large indexes – and requires information on the metrical structure. An approach to use more musical intelligence has been taken by Melucci and Orio (2004) by indexing musical phrases. We chose to use motifs rather than phrases, because motifs are shorter and not as specific to a piece and therefore better suited for building an index.

[1] http://www.themefinder.org/
[2] http://www.mudcat.org/AboutDigiTrad.cfm
[3] http://www.interactivemusicnetwork.org/mpeg-ahg/

2 ARCHITECTURE

The approach taken here is to index motifs using their contour classes, thus reducing the number of entries by joining similar motifs. Motif classes have been used for a similarity based search using representation at different degrees of abstraction by Melucci and Orio (2004), but that approach is insufficient to reflect the musical similarity of melodies, because it means introducing or reducing subdivisions of classes. However, similarities across the highest level of class boundaries cannot be detected. To broaden the search, here the similarity of different contour classes is utilized for fault tolerant and incremental search.

Especially on large databases, a sophisticated measure of similarity may be useful but computationally too expensive to perform on all pieces. These measures can be utilized in our system as a second-level search on the result set of the first search based on motif classes.

Before one can search a database of pieces, an index is generated with the following steps:

1. segmentation of the pieces
2. contour analysis of the motifs
3. indexing of motif contours

Once the index is created, melodies can be searched for. This is done in five steps:

1. segmentation of the query
2. contour analysis of the motifs
3. search for the motif classes
4. calculating similarity of the motif classes
5. fine search on the results

3 INDEX GENERATION

3.1 Segmentation

The segmentation divides a melody into motifs as in figure 1. It is the first step in processing and is important for the quality of the results. The more the segments form musically meaningful motifs of the melody, the better search results can be expected. To enable experiments on this, the actual segmenter module is exchangeable. For efficiency and simplicity a simple segmenter has been used in these first experiments that sets the boundaries at the local maxima of note distances. This approach has been applied successfully (Cambouropoulos, 2001), but using a more sophisticated segmenter could improve search results (see Weyde, 2002).

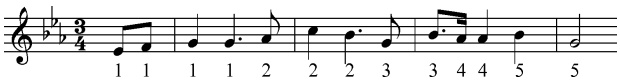

Figure 1: Segmentation of the melody "Der Mai ist gekommen" (German folk song).

3.2 Contour Analysis of Segments

Contour information is stored for two aspects of a segment: its rhythmic values and its pitches. The motif is coded using a diastematic index. This means that only three values per dimension are used for the transition between two notes: whether a note is longer, shorter, or of equal length and whether a note is higher, lower, or of same pitch. This can be represented as a character string $s(M)$, where $M = n_1, \ldots, n_m$ is a melody, $L(n_i)$ the length of note n_i and $P(n_i)$ its pitch, and $+$ denotes string concatenation:

$$s(M) = \begin{cases} s(n_1, \ldots, n_{m-1}) + k(n_{m-1}, m_n) & \text{if} \quad m \geq 2 \\ \text{“”} & \text{if} \quad m < 2 \end{cases} \tag{1}$$

$$k(n_1, n_2) = \begin{cases} \text{“U”} & \text{if} \quad P(n_1) < P(n_2) \\ \text{“E”} & \text{if} \quad P(n_1) = P(n_2) \\ \text{“D”} & \text{if} \quad P(n_1) > P(n_2) \end{cases} \tag{2}$$
$$+ \begin{cases} \text{“L”} & \text{if} \quad L(n_1) < L(n_2) \\ \text{“E”} & \text{if} \quad L(n_1) = L(n_2) \\ \text{“S”} & \text{if} \quad L(n_1) > L(n_2) \end{cases}$$

3.3 Indexing of the Segment Contours

A contour string $K = (k_1, \ldots, k_n)$ is converted into an integer number using the bijective function e:

$$e(K) = \begin{cases} e(k_{1\ldots n-2}) \cdot 2^4 + b(k_{n-1}, k_n) & \text{if} \quad n > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$b(k_1, k_2) = \begin{cases} 0100b & \text{if} \quad k_1 = \text{“U”} \\ 1000b & \text{if} \quad k_1 = \text{“E”} \\ 1100b & \text{if} \quad k_1 = \text{“D”} \end{cases} \tag{4}$$
$$+ \begin{cases} 01b & \text{if} \quad k_2 = \text{“L”} \\ 10b & \text{if} \quad k_2 = \text{“E”} \\ 11b & \text{if} \quad k_2 = \text{“S”} \end{cases}$$

s allows to code motifs up to a length of 9 notes in a 32-bit integer. This is below the theoretical limit of 10 notes[4], but it is sufficient because musically meaningful motifs do not exceed this length due to limits of perception (see Miller, 1956; Swain, 1986). If a segmenter produces segments longer than 9 notes, these notes will be cut off in the current implementation.

The contours of the example in figure 1 would be UEULEL $= 0110010110001b = 1625$, ULDL $= 01011101b = 93$, UL $= 0101b = 5$, EL $= 1001b = 9$ and DL $= 1101b = 13$. This coding as integer values allows efficient indexing of pieces by motif classes.

4 SEARCH PROCEDURE

The search is carried out in two stages: a rough search that uses a similarity value based only on the motif classes, and a fine search that elaborates on the result set.

4.1 Rough Search

Firstly the query melody is segmented and the motif contours are classified as described above. Then the pieces are rated for their similarity and relevance to the query.

4.1.1 Motif Class Similarity

The local similarity of two motif classes k, l is calculated as

$$lsim(k, l) = \frac{1}{d(k, l) + 1}, \tag{5}$$

where $d(k, l)$ is the Levenshtein distance between contours (see Gilleland 2004). It calculates the minimum number of changes needed to transform contour S into contour Q. A change is either the insertion, the deletion or the change of a transition as described in section 3.2.

4.1.2 Weighting

The contour classes of the query and of pieces in the database are weighted based on their frequency in the database and in the melodies as in standard text information retrieval (van Rijsbergen, 1979):

$$w_{S,l} = \frac{v_{S,l}}{\sqrt{\sum_k v_{S,k}}}, \tag{6}$$

with

$$v_{S,l} = \begin{cases} f(S, l) \cdot \log\left(\frac{N}{n_l}\right) & \text{if} \quad n_l \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

[4] Rhythm and pitch can each take 3 values yielding 9 combinations per note transition plus 1 if no note is present. Therefore the number of possible values for a motif of length l is 10^{l-1}. The maximal length of motifs that can be represented is therefore $\lfloor \log_{10}(2^{32}) + 1 \rfloor = 10$.

where $f(S,l)$ is the frequency of motif class l in melody S, N is the total number of pieces in the database, and n_l is the number of pieces containing a motif of class l. Imposing a penalty on common motif classes introduces a relevance aspect based on the pieces in the database, in addition to the concept of similarity between two melodies. The weights $w_{S,l}$ are normalized to make the values independent of the lengths of the S.

4.1.3 Similarity of Melodies

For the overall rating of a piece and in relation to a query, the most similar motif class in the piece is determined for every motif class present in the query, yielding a set L_{max} of pairs (k,l) of motif classes. The rating sim of a query Q and a piece S is then calculated as

$$sim(Q,S) = \sum_{(k,l)\in L_{max}} lsim(k,l) \cdot w_{Q,k} \cdot w_{S,l} \quad (8)$$

The sim measure yields values between 0 for completely different and 1 for identical melodies. The value 1 is also reached by melodies sharing the same distribution of motif classes. We did not encounter such a case in the experiments, but this possibility makes it especially useful for large databases to have a fine search that enables differentiation between such melodies.

4.2 Fine Search

The fine search takes the results of the rough query and performs comparisons, that are based on their the actual notes instead of the contours, which may be computationally more expensive. Currently two different fine search methods are implemented. They calculate similarity measures on the motifs, which are multiplied with the sim value. One method returns 1 only if the two compared segments are identical except for transposition and tempo changes, the second uses the *CubyHum* algorithm (Pauws, 2002), which is an edit distance variant specialized for melody retrieval.

5 EVALUATION

The described system has been implemented and tested using the MUSITECH framework (see Gieseking and Weyde, 2002).

5.1 Quality

For a first quality assessment we used small subsets of ground truth data from the experiments of Typke et al. (2005) and Müllensiefen and Frieler (2004) giving correlations of 0.59 and 0.36 between our sim values and the ground truth, which we see as a good result for a rough search. The currently implemented fine search improved the correlation only by up to 0.03, which indicates that here is a need for a different method.

5.2 Complexity

The time needed to compute the motif classes only depends on the length m of a melody, thus taking $O(m)$ time. A database containing N pieces of average length m needs $O(m \cdot N)$ time to be indexed. These values depend only on the pieces in the database, and therefore only need to be computed once for each piece. The amount of space required for the index also depends on the number of motif classes in a melody, which means at most linearly on the length of the melody, and the size of the database. So the space requirement for the index is also $O(m \cdot N)$.

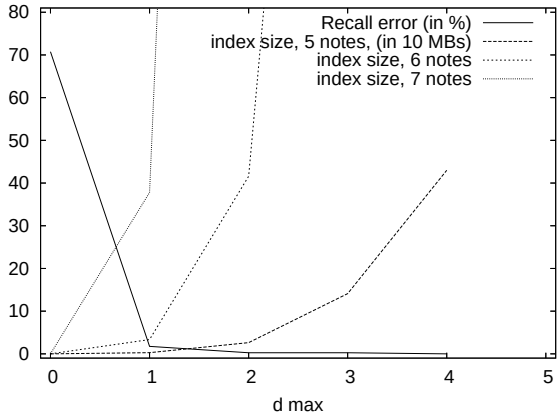

Figure 2: Recall error and index size depending on d_{max} and maximal motif length.

The time requirement for a naive implementation of the rough search is $O(q \cdot m \cdot N \cdot l^2)$ where q is the length of the query, since every motif class of the query must be compared to each motif class of every piece in the database. l^2 is the complexity of calculating the $lsim$ values, where l is the length of the motifs.

The fine search uses more time for each comparison, for example when using the CubyHum algorithm, it uses $O(q^2 \cdot m^2 \cdot R)$ time, where R is the size of the result set.

5.3 Optimization

The naive implementation of the rough search is too slow to be used on large databases, but it can be optimized using the opportunities provided by the motif classes. It is possible to perform much of the computing in advance and to use inverted files, reducing the time needed to search a potentially large database with the trade-off of higher storage demands.

Since the number of motif classes is limited, $lsim$ values of classes can be calculated beforehand, independently of any actual data. If the similarity for every pair of classes is stored, the size of the index then depends on the length of the motifs as $(\sum_{k=1}^{l_{max}} 9^{k-1})^2$ where l_{max} is the maximal length of the motifs. Judging from the literature, a motif length of 7 is sufficient, probably less according to Swain (1986). Yet storing the $lsim$ value for each combination would result in over a terabyte of data which is out of the scope of current computers. The index size can be reduced by indexing only those pairs of motif classes, where the $lsim$ value is above a threshold $lsim_{min}$, resp. $d(k,l)$ below a threshold d_{max}. This reduces reduces the size of the $lsim$ index and the number of matches for which a sim value has to be calculated. The trade-off is a loss in recall, i.e. some of the pieces be-

low the threshold might have received a good final rating. We computed the loss of recall of the rough search on a set of almost 4000 MIDI files from the Digital Tradition database[5] and the size of the index for different values of d_{max} and l_{max} as shown in figure 2. The recall error was calculated as the portion of pieces in the top 10% that is excluded by the threshold (which is in this case identical to the precision value). It drops sharply from d_{max} values 0 to 1 from over 70% to 1,75% and to 0,25% at $d_{max} = 2$, while the index size grows exponentially with at d_{max} and l_{max}. A good compromise can be found at $d_{max} = 1$ and $l_{max} = 6$ where the index size is below 50 MB which can easily be held in RAM. This method also supports incremental search, first using $d_{max} = 0$ and then $d_{max} = 1$.

The class frequencies can be stored along with the melodies in the motif index, as can the inverted frequencies of classes over the database, but they must be updated at every change of database content. Using this method, the whole rough search takes $O(q \cdot R_t)$ time where R_t is the size of the result set after the threshold. The constant factor is also reduced, because of the lookups of the $lsim$ values. The size of R_t depends on the data and the queries and was with $d_{max} = 1$ and $l_{max} = 6$ at about 50% of the database size in our experiments. The size of R_t can be influenced by a second threshold d_{prod} on the product $w_{Q,k} \cdot lsim(k,l) \cdot w_{S,l}$. Since $w_{Q,k}$ is available at the time of the search and both other values are precomputed, it can be used to reduce R_t before computing the sim values.

6 CONCLUSIONS

The use of motif classes brings two advantages to melody retrieval. It makes the matching of the query with melodies in the database independent of the position and order of motivic material used. It therefore makes the extraction of themes unnecessary because the length of pieces it not relevant. It also captures musical relations, that are not taken into account by approaches based on the edit distance of whole melodies.

It also offers methods for efficient and incremental searching through the use of indexes and similarities. The use of precalculated similarities of motif classes allows very efficient and musically adequate incorporation of inexact matches.

Directions for further research include testing on further ground truth data, experimenting with fine search methods to improve the end results, tests with more elaborate segmenters, and the investigation of methods to control the size of the result set R_t in oder to optimize search times and result quality.

REFERENCES

E. Cambouropoulos. The local boundary detection model (lbdm) and its application in the study of expressive timing. In *Proceedings of the International Computer Music Conference 2001*, pages 290–293, Havanna, Cuba, 2001.

M. Clausen, F. Kurth, and R. Engelbrecht. Context-based retrieval in midi and audio. In D. Fellner, N. Fuhr, and I. Witten, editors, *ECDL Workshop: Generalized Documents*, Darmstadt, 2001.

M. Gieseking and T. Weyde. Concepts of the musitech infrastructure for internet-based interactive musical applications. In C. Busch, M. Arnold, P. Nesi, and M. Schmucker, editors, *Proceedings of the Second International Conference on WEB Delivering of Music (WEDELMUSIC 2002)*, pages 30–37, Darmstadt, 2002. IEEE / Fraunhofer IGD.

M. Gilleland. Levenshtein distance, in three flavors, 2004. URL `http://www.merriampark.com/ld.htm`.

M. Melucci and N. Orio. Combining melody processing and information retrieval techniques: methodology, evaluation, and system implementation. *J. Am. Soc. Inf. Sci. Technol.*, 55(12):1058–1066, 2004.

G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, 1956.

D. Müllensiefen and K. Frieler. Cognitive adequacy in the measurement of melodic similarity: Algorithmic vs. human judgments. *Computing in Musicology*, 13:147–176, 2004.

S. Pauws. Cubyhum: A fully operational query by humming system. In M. Fingerhut, editor, *ISMIR 2002 Conference Proceedings – Third International Conference on Music Information Retrieval*, Paris, 2002. IRCAM – Centre Pompidou.

H. Riemann. *System der musikalischen Rhythmik und Metrik*. Breitkopf und Härtel, Leipzig, 1903.

J. P. Swain. The need for limits in hierarchical theories of music. *Music Perception*, 4(1):121–148, 1986.

R. Typke, M. den Hoed, J. de Nooijer, F. Wiering, and R. C. Veltkamp. A ground truth for half a million musical incipits. In *Proceedings of the 5th Dutch-Belgian Information Retrieval Workshop (DIR) 2005*, pages 63–70, Utrecht, 2005.

C. J. van Rijsbergen. *Information Retrieval*. Butterworth, London, 1979.

T. Weyde. Integrating segmentation and similarity in melodic analysis. In K. Stevens, D. Burnham, G. McPherson, E. Schubert, and J. Renwick, editors, *Proceedings of the International Conference on Music Perception and Cognition 2002*, pages 240–243, Sydney, Australia, 2002.

[5] `http://sniff.numachi.com/~rickheit/dtrad/`.