FAST MUSIC RETRIEVAL USING POLYPHONIC BINARY FEATURE VECTORS

Hidehisa Nagano, Kunio Kashino and Hiroshi Murase

NTT Communication Science Laboratories, NTT Corporation
3-1 Morinosato Wakamiya, Atsugi-shi, Kanagawa, 243-0198 Japan
{nagano, kunio, murase}@eye.brl.ntt.co.jp

ABSTRACT

We propose a method for retrieving similar music from a polyphonic-music audio database using a polyphonic audio signal as a query. In this task, we must consider similarities among polyphonic signals of the music, and achieve quick retrieval. Therefore, we first introduce the polyphonic binary feature vector to represent the presence of multiple notes. This feature is suitable for the search based on the similarities among polyphonic audio signals. Then, we propose a new search method, which is quicker than the exhaustive use of DP matching. The search is accelerated using a "similarity matrix" to limit the search space. Experiments using a test database containing 216 music pieces show that the search accuracy of the proposed feature is 89%, which is approximately 26% higher than that of the conventional spectrum feature. It is also shown that the new search method retrieves similar music without significant accuracy degradation as well as the exhaustive search does and the computational complexity of the new search method is about 1/4 that of exhaustive search.

1. INTRODUCTION

This paper proposes a music retrieval method based on polyphonic music similarity of audio signals. Polyphonic music search from a polyphonic audio signal database (DB) using a polyphonic audio signal as the query extends the applicability of music retrieval. For example, it will enable us to retrieve similar music from audio DBs using music pieces on CDs as queries, which are polyphonic signals. In addition, we aim at retrieving not only the same segments but also "similar" ones, such as re-takes, the music played on other instruments, with a different tempo, or transposed.

Music retrieval is an important application of information retrieval and several search methods for it have been proposed [1, 2, 3, 4]. These methods can be divided into two groups according to their objectives. One aims at retrieval of segments that are almost the same as the query by a signal level comparison [1]. In this case, audio signal features, such as the spectrum feature, are used for the search. The other group aims at retrieving segments "similar" to the query in some way, such as having the same melody or a rearrangement of the music. Such methods require fast retrieval of similar segments using musical queries other than audio signal features. For retrieval based on melody similarity, Ghias et al. proposed monophonic melody matching using a melodic contour that represents the melody as strings of relative pitch ('U', 'D' and 'S') and showed its effectiveness for retrieval with queries by humming [2]. The melodic contour is too simple to represent the complete melody, but it serves as a valid feature for the classification of the monophonic melodies and is efficient for music retrieval based on monophonic melody matching. So far, most related works have also been based on monophonic melody matching.

However, those existing methods are not directly applicable to our music retrieval task. Audio signals of music pieces considerably differ, even if they are the same music in a title, due to re-takes, performance speed, and changes of instruments, for example. Therefore, features directly related to the audio signals are not always appropriate for retrieval. Moreover, applying monophonic matching to our task would require the complicated extraction of multiple monophonic melody lines from the polyphonic audio signal. In addition, time consuming melody matching among multiple melody lines must be considered. Therefore, we propose a polyphonic binary feature vector as a simple feature representation of the polyphonic audio signal that is suitable for classification of polyphonic music signals and their matching. Like Ghias's melodic contour, the objective of this feature is not the complete representation of the music but the classification of the music for retrieval. This feature vector is a simple representation of the polyphonic chord and suitable for quick retrieval. We also, then, propose a fast search method for retrieval using this feature.

The rest of this paper is organized as follows. Section 2 overviews the retrieval scheme. Section 3 proposes the polyphonic binary feature vector, and Section 4 introduces the fast search method. Section 5 then shows some experimental results. Finally, Section 6 concludes the paper.

2. OVERVIEW

Fig. 1 overviews the search flow, where all segments similar to the query signal on the stored signal are being detected. The query signal and the stored signal are polyphonic acoustic signals of music. First, we extract feature vectors from the frequency spectrum and prepare them as feature vector strings for searching. Each component (symbol) of the feature vector string corresponds to a feature vector. Then, we search for all the substrings of the length w that are similar to the query feature vector string. The length w is assigned as the window width on the stored feature vector string. To cope with temporal stretching and shrinking of signals, we measure the similarities between feature vec-

0-7803-7304-9/02/$17.00 C2002 IEEE

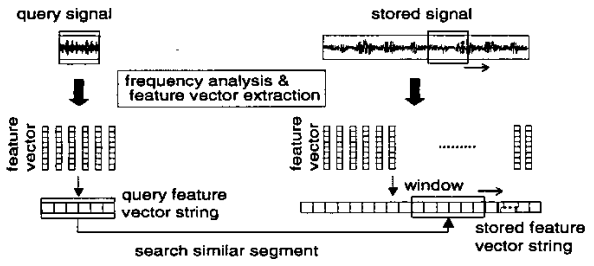


Figure 1: Overview of searching.

tor strings by DP matching based on the similarities among feature vectors. Formally, we search for all substrings of the stored feature vector string with the length w such that its normalized similarity with the query feature vector string given by DP matching is larger than the search threshold s_t.

3. POLYPHONIC BINARY FEATURE VECTOR

As a simple feature representation suitable for a search like melodic contour but also applicable to polyphonic signal, we propose the polyphonic binary feature vector (hereafter, the PBFV). The PBFV is a simple representation of the presence of notes from multiple instruments by each binary vector element and suitable for the search based on similarity of the presences of the notes.

During the extraction of the PBFV, beat tracking and harmonics elimination are performed. A PBFV is extracted for each beat interval. Beat tracking is done to shorten the feature vector string. In addition, it is also expected that this beat tracking absorbs the shrinking and stretching of the audio signal. Harmonics elimination is adopted, since different harmonics yielded by different instruments and dynamics can drastically alter the frequency spectrum. However, the detailed processes of beat tracking and harmonics elimination are not the focus of this paper. Therefore, we explain these processes roughly for want of space.

3.1. Audio feature extraction

First, we process audio files through a band pass filter bank, and construct a frequency-power spectra profile from the audio signal of the music. The bank consists of 336 filters that are equally spaced in the log-frequency axis from 75 to 9600 Hz. The analysis frame length currently used is 44 ms with a shift of 11 ms.

3.2. Beat tracking

Next, we extract the beats based on the audio signal changes [5]. The beat positions are assigned at points where the acute power change concentrates in the spectra.

3.3. Harmonics elimination

Local peaks of the power in the frequency axis are marked on each beat interval on the spectra. The extracted peaks are subsequently assigned to the nearest corresponding frequency on the 12-note Western scale, and in this way a list of "component presences" is constructed for each beat interval.

Harmonics elimination is done by scanning the components on the component presence list from low frequency to high. The components are deemed to be fundamental only if a certain number of their expected harmonics are also present on that beat interval, and other components are eliminated.

3.4. Binary feature vector representation

From the component presence list, we extract the PBFV. Here, the components in the component presence list in a beat interval are placed into a 12-element binary feature vector, the PBFV, that represents the notes on the Western musical scale. Each slot in these vectors holds information about note-presence (value of 1) or note-absence (value of 0). For example, if the components in the component presence list are A (220Hz), E (330Hz), A (440Hz) and C (523Hz), then the components recorded will be simplified to "A, C, E" and will yield the binary vector "000010001001", where the binary vector presents the presence of each note in the order "G#GF#FED#DC#CBA#A." Each PBFV is assigned to a unique symbol. In this way, these extracted symbols compose the the feature vector string.

The similarity between the PBFV $\mathbf{x} = (x_1, x_2, \ldots, x_{12})$ and $\mathbf{y} = (y_1, y_2, \ldots, y_{12})$ is defined as

$$\frac{\sum_{k=1}^{12} u_k}{\sum_{j=1}^{12}(x_j + y_j)}, \tag{1}$$

where $u_k = 2$ if $x_k = y_k = 1$, else $u_k = 0$. If $\mathbf{x} = \mathbf{y} = \mathbf{0}$, the similarity is 1.

This PBFV can represent polyphonic note presence compactly, and the similarity is suitable for the polyphonic music comparison. In addition, transposed music can be retrieved simply by shifting the PBFVs of the query.

4. FAST SEARCH METHOD

Here, we propose a fast method for the search overviewed in Section 2. The conventional exhaustive search method, in which DP matching with all substrings of the stored feature vector string is done while shifting the window by a symbol (feature vector), could be used. However, for quick retrieval, we propose a faster search method. The main idea for acceleration is to match only the substrings of the stored feature vector string that include symbols similar to those of the query feature vector string while reducing the computational costs of DP matching. As explained in the rest of this section, the proposed search method guarantees a search result identical to that obtained by the exhaustive search when the same symbol similarities are used, and achieves even faster search when the "sparse" definitions of the similarities are used.

First, we introduce a similarity matrix for symbols of feature vectors. The value of the (u, v) component of the similarity matrix is the similarity between symbols u and v. This similarity matrix is also used in the exhaustive search method to look up the similarity between symbols. We further accelerate the search by making the similarity matrix sparse. Here, we call the matrix sparse when almost all values of the matrix components are 0. The matrix is made sparse, for example, by threshold operation on the matrix components. Here, let $\mathbf{T} = [c_T(1), c_T(2), \ldots, c_T(n)]$ be the stored feature vector string, and $\mathbf{P} = [c_P(1), c_P(2), \ldots,$

$c_P(m)$] be the query feature vector string. Again, w is the length of the window.

Fig. 2 overviews the proposed search method. First, the window is moved to the place that includes a symbol similar to one of **P** as shown in Fig. 2(a). Then, the stored feature vector and the substring in the window are matched as shown in Fig. 2(b). Fig. 2(b) shows the graph used for DP matching, where the horizontal axis and the vertical axis correspond to **P** and the substring of **T** in the window, respectively. The borderlines adjust the survival paths and avoid immoderate correspondences of symbols. Especially advantageous is that, in this matching, the survival paths to white points in Fig. 2(b) are not calculated and their similarities are assumed to be 0. Thus, the computational cost is reduced compared with conventional DP matching. Now, we describe the proposed search method in detail.

Proposed Search Method

(1) For each kind of symbol, check where it appears on **T**.

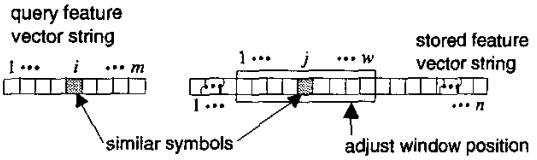
(2) For d from 2 to z, repeat (3). Here, z is the greatest integer less than $(1 - s_t)(m + w) + 2$.

(3) For i from 1 to $\min(m, d-1)$, let $j = d - i$. If the lattice point (i, j) is between the borderlines in Fig. 2(b), adjust the window to the places on **T** so that the similarity between $c_P(i)$ and the j-th symbol of the window is larger than 0 as shown in Fig. 2(a). If the adjusted place has not been matched, the substring and **P** is matched as shown in Fig. 2(b) and this place is registered as matched. After the match, if the similarity of the adjusted place is larger than the s_t, this place is detected as a similar segment. □□

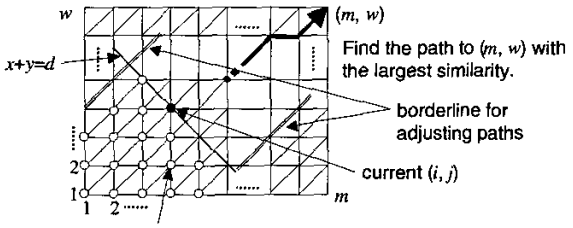
In Step (3), matching is performed when the window is adjusted to the unmatched place. Therefore, the similarity between $c_P(x)$ and the y-th symbol of the substring is 0 if the values of x and y have previously been used as the values of i and j. Then, the similarity of such correspondence of symbols, i.e., the similarities of the paths to the white points in Fig. 2(b) is 0 and not calculated.

In addition, we reduce the computational cost by the "omission of hopeless paths." If the upper bound value of similarities of the paths through a point, which is given by the similarity of the survival path arriving at this point and the length of the rest the paths, is not larger than the search threshold, further search on paths through this point is omitted.

For the evaluation of the computational complexity, we measured the number of points whose survival path must be calculated. We call such points computing points and the computation of a survival path at a point a path computation. The path computation at a point selects the path with the largest similarity from the paths that connect adjacent points to that point, and the complexity of this computation is constant. Assuming that the number of points between the borderlines in Fig. 2 is c, the number of path computations of the conventional DP matching is also c and the total number of the path computations for the conventional exhaustive search is $c(n - w + 1)$. For the proposed search method, the number of path computations in



(a) Adjustment of window position



(b) Matching with computational reduction

Figure 2: Overview of proposed search method.

the worst case is the same, but the number in the average case is $(c - \sum_{i=1}^{c} p_d^i)(n - w + 1)$, assuming the probabilities that the similarity of the two symbols are 0 are constant p_d. Here, the reduction of path computations by bounding the value of d with z in Step (2) and omitting hopeless paths in Step (3) is not taken into account, although the number would be even less if it were considered. From this formula, we can see that the sparser the similarity matrix is, the more efficient the proposed method becomes.

Until now, fast algorithms for "exact string matching" and "approximate string matching" have been proposed [6]. However, the proposed search method aims to search "similar" strings that stretch and/or shrink. Thus, it utilizes DP matching based on the similarities among symbols. These are the characteristics of our approach.

5. EXPERIMENTS

We performed two search experiments: one to evaluate the accuracy of the search using the PBFV, and the other to evaluate the efficiency of the proposed search method in reducing computational cost.

5.1. Evaluation of features

First, we compared the search accuracy of the PBFV with the conventional spectrum feature. For the comparison, we performed trial retrieval of similar music pieces from a DB of audio signals. The test DB contained audio signals of 216 polyphonic music pieces of various genres, such as pop, classical, and instrumental. It was totally about 480-min long. As query signals, we chose 15 segments of audio signals of three-part ensemble music so that the lengths of the query feature vector strings of the PBFV came to 70, and we performed a trial search for each query. The average length of the query signals was about 19 s. The DB includes the eight similar pieces having the same title as each query. The variations of these eight pieces are the original, a transposition, a change in tempo, a change of instruments, and re-takes of these. These variations were actually performed by musicians and recorded. The instruments used for these

Table 1: Search accuracy of features.

Feature	Spectrum	PBFV (proposed)
Accuracy	63%	89%

performances were piano, flute, oboe, bassoon, violin, and cello. In the change of instruments, one or two parts were played on other instruments. The query signal was chosen from the segments of the piece of the original. For each trial, we assumed that these eight pieces with the same title are to be retrieved, which we call the target pieces.

For the search, we extracted two kinds of feature vector strings. One employed the PBFVs and the other employed the spectrum feature. Then, we performed the search in Section 2 by the conventional exhaustive search method for both features. The spectrum feature was extracted with the filter bank in Section 3.1, but for a reasonable search time, it was extracted every 100 ms with the analysis frame length of 400 ms. Then, an LBG-based VQ with code book size of 4096 was employed for the comparison with the PBFV. We let the similarities of the spectrum features be 1 if they had the same VQ code and 0 otherwise. For the PBFV, the similarities defined in Eq. (1) were used, and 12 kinds of shifted queries were searched. In this experiment, for the PBFV, the total length of the stored feature vector strings was 94744. In this search experiment, we set the window width w the same as the length of the query string, and in DP matching, the survival path was adjusted so that $|x - y| < 0.06m$ for any point (x, y) on the path.

The search accuracy was evaluated by the average values of precision and recall rates when they were adjusted to be the same by choosing the search threshold for each query. The precision rate and recall rate are defined as #(correct retrieved pieces)/#(retrieved pieces) and #(retrieved target pieces) /#(target pieces), respectively. A piece is retrieved if at least one similar segment is detected on it.

Table 1 shows the results. Accuracy is the average accuracy of 15 retrievals. We can see that the average accuracy for the PBFV is 89%, approximately 26% higher than that for spectrum feature.

5.2. Evaluation of search methods

Another experiment was performed to compare the computational complexities and accuracy of the search method proposed in Section 4 with those of the exhaustive search method. In this experiment, the DB, the queries and the target pieces were the same as in Section 5.1. We performed these retrievals using both search methods.

For evaluation of the computational complexity, we evaluated the number of the path computations needed for the search by a query, when the precision and recall rates were the same.

For both methods, the PBFVs were used, and 12 kinds of shifted queries were searched. For the exhaustive search method, the similarity matrix with the similarities of PBFVs defined as in Eq. 1 were used. For the proposed method, we used a sparse similarity matrix obtained by making the component value 0 when it was smaller than the threshold value of 0.85. The window width setting and the adjustment of the survival paths are the same as in Section 5.1.

Table 2: Search method evaluation results.

Method	Exhaustive Search	Proposed Method
Accuracy	89%	86%
#path computations	457962240	119386645

Table 2 summarizes the results. Accuracy is the average accuracy of 15 retrievals. The #path computations shows the average value of the numbers of path computations. The proposed method can retrieve similar music with about 1/4 the path computations of the exhaustive search with only a small amount of accuracy degradation due to the use of a sparse similarity matrix. The "omission of the hopeless paths" described in Section 4 can be employed for the exhaustive search, but the #path computations in that case was 248848157, which is about twice as large as that for the proposed method.

6. CONCLUSIONS

We proposed the polyphonic binary feature vector (PBFV) as the feature suitable for retrieval of similar polyphonic music. In addition, a fast search method for this retrieval was proposed. Experimental results show that average retrieval accuracy using the PBFV is 89%, approximately 26% higher than that of the spectrum feature and the average computational complexity of the proposed search method is about 1/4 that of exhaustive search.

For future work, we plan to confirm the effectiveness of our method with a larger DB and various kinds of music. We will also investigate the construction of a similarity matrix for faster and more accurate retrieval.

7. ACKNOWLEDGEMENTS

The authors thank Brian Toth, Dr. Kenichiro Ishii, and Dr. Noboru Sugamura for their help and encouragement.

8. REFERENCES

[1] K. Kashino et al., "Time-series Active Search for Quick Retrieval of Audio and Video," in *Proc. of ICASSP-99*, 1999.

[2] A. Ghias et al., "Query By Humming: Musical Information Retrieval in An Audio Database," in *ACM Multimedia '95*, 1995.

[3] L. A. Smith et al., "Sequence-Based Melodic Comparison: A Dynamic Programming Approach," in *Melodic Similarity : Concepts, Procedures, and Applications*, W. B. Hewlett et al., Eds. MIT Press, 1998.

[4] N. Kosugi et al., "Music Retrieval by Humming: Using Similarity Retrieval over High Dimensional Feature Vector Space," in *Proc. of IEEE PACRIM'99*, 1999.

[5] K. Kashino, *Computational Auditory Scene Analysis for Music Signals*, Ph.D. thesis, Univ. of Tokyo, 1994.

[6] R. Baeza-Yates et al., *Modern Information Retrieval*, Addison Wesley, 1999.