AUTOMATIC MAIN MELODY EXTRACTION FROM MIDI FILES WITH A MODIFIED LEMPEL-ZIV ALGORITHM

Hsuan-Huei Shih, Shrikanth S. Narayanan and C.-C. Jay Kuo

Integrated Media Systems Center and Department of Electrical Engineering-Systems

University of Southern California, Los Angeles, CA 90089-2564

Tel: (213) 740-8386, Fax: (213) 740-4651, E-mail: {hshih,shri,cckuo}@sipi.usc.edu

ABSTRACT

A dictionary-based approach for extracting repetitive patterns in music aimed at music feature extraction and indexing for audio database management is proposed. In this system, segmentation is achieved based on the tempo information and a music score is decomposed into bars. Each bar is indexed and a bar index table is built. Then, an adaptive dictionary-based compression algorithm known as Lempel Ziv 78 (LZ-78) is applied to the bar-represented music scores to extract repetitive patterns. Finally, pruning is performed to this dictionary to remove non-repeating patterns and combine shorter repeating patterns into a longer repeating pattern. The LZ78 algorithm is slightly modified to achieve better results in the current context. Experiments are performed to MIDI files, and the proposed algorithm has demonstrated an excellent performance.

1. INTRODUCTION

Techniques for image and video feature extraction, indexing and retrieval have received a lot of attention recently in image and video database applications. In contrast, a relatively small amount of effort has been put in audio feature extraction and indexing. Audio database management finds applications in music archiving, special effect sound search in audio editing, etc. Audio is also an integral part of multimedia database, and often contains useful information for effective multimedia search. Multimedia database management with multimodal information such as audio, video and text, is a recent trend. A better understanding of audio features is an essential step towards a complete multimedia database management system.

Some work has been done in audio database organization, indexing and query. Chen, *et al.* proposed a pat trees approach to index melodies in [1], where pat trees were built by "chord". Ghias, *et al.* used coarse melody contours as a key to query in a music database [2]. Furthermore, Chen, *et al.* proposed the string-join approach [3] and the correlatve matrix approach [4] to find repetitive pattern in music. In the former approach, they repeatedly joined shorter repeating patterns to form a longer one. In the latter approach, they lined up a melody string in the x- and y-axis directions to form a correlative matrix and used it to find repeating patterns. McNab, *et al.* [5][6] used interval contours for interactive music retrieval. Tong and Kuo [7] considered a hidden Markov model (HMM) method to model special effect sounds for content-based audio query.

In this work, we focus on the extraction of repetitive patterns in the main melody of a given music piece. Repetitive patterns can be used in organizing music database, music database indexing, and as an important feature for content-based retrieval in music database. Moreover, they can be used for analyzing music database or even music composers' habit. It is believed that people are more sensitive and receptive to certain salient portions in a piece of music. In this work, we assume that repeating melody represents a salient part since composers usually compose a piece of music with certain small pieces of melody that tend to be repeated throughout the music piece. Therefore, repeating melodies are usually easily memorized by people. If a piece of music is written in music score form, repeating melodies in a piece of music are repeating patterns of notes in its music score. We use a dictionary approach to find these repetitive patterns based on the classic work of Lempel and Ziv [8][9].

The rest of this paper is organized as follows. In Section 2, the proposed algorithm is described in detail. Experimental results are given in Section 3 and concluding remarks are presented in Section 4.

2. PROPOSED ALGORITHM

The input to our proposed system is a piece of music consisting of numerical music scores. Thus, other music forms such as the sound wave have to be converted to numerical music scores first. The proposed algorithm is detailed below.

2.1 Music Decomposition and Bar Indexing

The bar (rather than the note) is used as a basic unit in our system due to the following considerations. First, the music note is a unit that is too fine to build a dictionary since there are too many notes in a piece of music and the complexity of the dictionary building process grows very quickly. Second, a note is adopted to preserve tempo information. In music scores, there are time signatures that are used to tell people the tempo of the underlying music, and a single piece of music may contain more than one time signature. In a music score, bars are used to group notes together according to a specified time signature. In our algorithm, bars are chosen to be the basic unit where a group of notes of the same time period are cascaded.

Usually, several bars form a repeating pattern. However, a repetitive pattern may not start from the beginning of a bar and stop at the end of a bar. In other words, they may start and stop at any note in a bar. For a given song, repetitive patterns tend to start and stop at fixed positions in a bar. Let us consider an example in which a repeating pattern appears twice. There could be one or two bars that contain the repeating pattern's starting note. The time-offsets of these two starting notes in their respective bars are often the same while music notes in the offset regions could be different. The same observation applies to the end point of a repeating pattern. The intermediate bars that lie

between the starting and the ending bars are exactly the same. The leading and ending bars of a repetitive pattern requires some special handling.

After decomposing a piece of music into bars, we should find a concise representation of each bar for further processing. To do so, a simple bar index table is generated. There are three attributes in this table: (1) a unique index number, (2) the bar pattern, and (3) the frequency of occurrence of this bar pattern. The index number is chosen to be the sequence of non-negative integers, i.e. 0, 1, 2, 3, etc. The bar pattern is the sequence of note's pitches which are expressed in terms of numbers. The bar index frequency is the appearance of each bar in a piece of music.

While the bar index table is being built, segmented music score is also being converted into a bar indexed music score which is replacing each bar with its corresponding index in the music score. To facilitate the process of matching two bars, we can extract attributes from the bar pattern and perform attribute matching to filter out unlikely candidates. For example, the number of notes in a bar is an attribute that can be easily exploited. Furthermore, we can record pitch values of consecutive notes in a bar while ignoring their durations and, at the same time, discard all rest notes to derive another attribute. For this attribute, rests at the start, in the middle, or at the end of a bar are treated the same in the bar matching process.

2.2 Lempel-Ziv 78 (LZ78) and Its Modification

Lempel-Ziv 78 (LZ78) is a lossless compression scheme that has been widely used in text compression. A dictionary of variable length is constructed and adaptively updated by LZ78 in parsing a sequence of symbols. Vocabularies in the dictionary will be added according to the processed data. In our system, input symbols are bars with an appropriate index number and vocabularies in the dictionary are sequences of bar indices. The idea of dictionary-based compression lies in that we should detect longer vocabularies and encode them with shorter codewords. This process turns out to be a powerful tool to find repeating patterns. The block diagram of LZ78 is shown in Figure 1 and explained below.
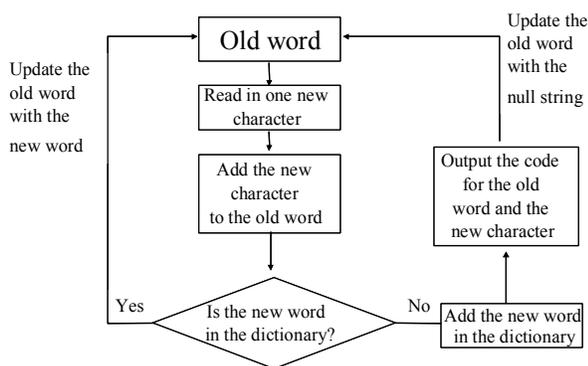

Figure 1. The block diagram of LZ78.

1. A word buffer and a dictionary are needed in LZ78, and they are both set to empty in the beginning. The old and new words as shown in Figure 1 are stored in the word buffer.
2. Read in one new symbol from input data. In our case, the symbol is a bar index.

3. Append the new symbol to an old word to form an updated word. Now, there is at least one character in the word buffer.
4. If the updated word is already in the dictionary, then it is treated as an old word and nothing is changed in the buffer. We return to Step 2. If the updated word is not in the dictionary, then we add this updated word to the dictionary as a new word, output the index of the old word (i.e. the updated word without the new character) in the dictionary, and empty the word buffer. After that, we go back to Step 2.

By using LZ78, to form a repetitive pattern of N bars requires that the pattern appears at least N times in the underlying music. If N is large, it could be difficult to get a long repetitive pattern. To overcome this difficulty, we pass the same music piece into the LZ-78 dictionary building system several times. According to our experiment, to pass the music piece repeatedly for 3 or 4 times will be sufficient to catch a repeating pattern of about 12 bars long.

Not every pattern in this dictionary is a part of repeating pattern, and pruning is needed to give the final result. Before a detailed discussion of the pruning process, let us comment on three issues related to dictionary building. First, the starting note of every repetitive pattern may not be aligned well with the extracted patterns in the dictionary. Second, many bars that are not repeating in the bar index table may be included in repeating patterns, since LZ78 process is applied to the same music more than once. Third, how to determine the end of the repeating pattern? It is difficult to find the end point in the pruning phase by using this dictionary. To address these issues, LZ78 has to be modified to get better results. The block diagram of modified LZ78 is shown in Figure 2.
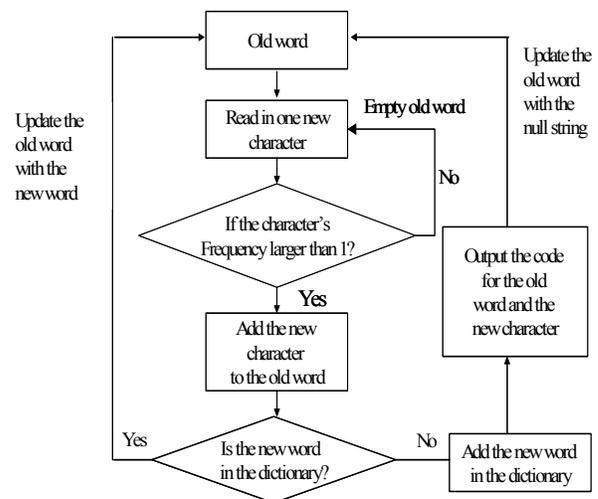

Figure 2. The block diagram of modified LZ78.

Since the frequency of each bar is known a priori before LZ78 is applied, we can use this information more effectively in LZ78. That is, for a bar with its frequency equal to one, we know that its index number should not be a part of a repeating pattern. The buffer in the LZ78 process should be forced to drop whatever it has when a non-repeating index is read in and start a new empty buffer again. Another modification is to record the "pattern

frequency" attribute in the LZ78 process. It is used to count the appearance of each pattern. The pattern frequency will be updated when the corresponding pattern is in the buffer (No matter the buffer is in an intermediate step, or it is in the final step while a new pattern is being added to the dictionary). With these two modifications, it solves the problems mentioned previously.

2.3 Pruning

There are five steps in the pruning phase.

1. A threshold of the minimum length of a repeating pattern is selected. We check if a pattern in the dictionary is shorter than the threshold. If it is shorter than the threshold, it is discarded. If it is longer and equal to the threshold, we put it in the candidate list.

2. Check patterns in the candidate list whether they are indeed repeating patterns in the bar-represented music score. Eliminate non-repeating patterns. This might happen because the bar frequency is the frequency of each individual bar's appearance in a piece of music and it is independent of appearance in time. Therefore, there may be a bar with its frequency more than one but appears in isolated places.

3. Eliminate proper subsets of a longer repeating pattern. If a repeating pattern in the candidate list is a proper subset of other repeating pattern, then remove it from the candidate list. Do this until no any repeating pattern is another repeating patterns' proper subset.

4. Join repeating patterns that have common parts. "Join" is an operation in relational algebra. It is defined as follows. Consider two operands, where one is the front operand and the other is the back operand. Each operand can be divided into two parts from any place in the middle of a pattern, and becomes left and right parts. Join will only work for any two patterns that have common sub-pattern, where the sub-pattern is both the right part of the front operand and the left part of the back operand. For example, A: {a b c}, B: {c d e}, C: {d e f}; A join B equal to {a b c d e}, and A cannot join C. Join two repeating patterns when this is possible.

5. Return the list of repeating patterns and order them as follows. The one with the highest occurrence is listed first. If two repeating patterns have the same number of occurrences, then the pattern of shorter length first. If they have the same length, the one appears earlier in the music piece first.

6. Finally, convert the bar indices to bar patterns by using the bar index table.

After pruning the system, we will obtain a list of repeating patterns that can be used in music database indexing or as extracted features for content-based audio query.

3. EXPERIMENTAL RESULTS

3.1 Experiment Setup and Example

We collected 200 MIDI files of the seventies' and eighties' pop music genre from the public domain to form our database. The system was implemented in C++ on an Intel PC. According to the time signature information contained in the MIDI file, we segmented numerical music scores into bars and

then applied our algorithm to the bar representation. For illustration, we will use the piece "Yellow Submarine" by Beatles as an example. Figure 3 shows the first nine bars of Yellow Submarine in music score and its numerical pitches, where the pitches are from the MIDI file, under each note. In a MIDI file, the ranges of pitches are within 0 to 255, so ASCII characters are used to replace numerical pitches in our experiment, and each note is looked as one byte symbol. After converting the music score to the numerical music score, the numerical music score is segmented into bars according to the time signature, which is 4/4 for Yellow Submarine. Then, the decomposed numerical music score is used to build a bar index table and converted into bar indexed music score base on the bar index table. Table 1 shows the entire bar index table of Yellow Submarine.
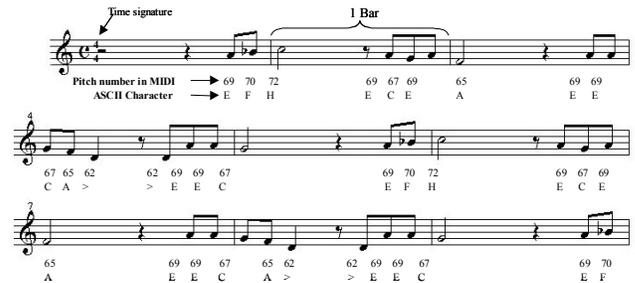

Figure 3. The First nine bars of the music score of "Yellow Submarine".

Index	Pattern	Freq.		Index	Pattern	Freq.
0	EF	1		6	C	2
1	HECE	10		7	HHHHJ	6
2	AEE	9		8	CCCCC	12
3	CA>>EE	3		9	AAAAA	3
4	CEF	6		10	AAAAAEF	3
5	CA>EE	6		11	A	1

Table 1. Bar index table for "Yellow Submarine"

The left part of Figure 4 shows the entire bar indexed music score. We applied the modified LZ78 to the bar indexed music score for 4 times. There are 28 patterns in the modified LZ78 dictionary, as shown in Figure 5. Let us set the threshold of the minimum length of a repeating pattern to 5 in this example. Finally, 24 patterns are eliminated in the pruning phase and four repeating patterns are left as our final result as shown in the right part of Figure 4.

3.2 Experiment Analysis

The number of repeating patterns in the resulting list varies with different songs. Some songs may have more repeating patterns while some less. The length of repeating patterns is not a constant either. The number of segmented bars of a piece of music is dependent on the length of the input music. But the bar index table of a piece of music will be dependent on the piece of music. In our system, we set only the minimum length since very short

repeating patterns are not important. Table 2 shows statistical information of tested songs. As seen in table 2, some repeating patterns are very long after pruning and some are unchanged. For example, Hotel California by Eagle has only two very long repeating patterns that are truly the two main melodies.

Title	NO. of bars	Bar index table Size	MLZ 78 dictionary size (patterns)	NO. of repeating patterns	Max. repeating pattern Length
Yellow Submarine	59	12	28	4	10
All I have to do is dream	55	27	8	2	5
500 miles	52	24	8	3	4
Hotel California	117	22	47	2	15

Table 2. Statistical information of tested songs

Bar indexed music score	List of repeating pattern
0 : 1 : 2 : 3 : 4 : 1 : 2 : 3 : 4 : 1 : 2 : 3 : 4 : 1 : 2 : 5 : 6 : 7 : 8 : 8 : 9 : 7 : 8 : 8 :10 : 1 : 2 : 5 : 4 : 1 :11 : 7 : 8 : 8 : 9 : 7 : 8 : 8 :10 : 1 : 2 : 5 : 4 : 1 : 2 : 5 : 6 : 7 : 8 : 8 : 9 : 7 : 8 : 8 :10 : 1 : 2 : 5 : 4 : 1 : 2	1. 1:2:5:6:7 2. 1:2:5:4:1:2 3. 1:2:3:4:1:2:3 4. 6:7:8:8:9:7:8:8:10:1:2

Figure 4. left: Bar index music of Yellow Submarine. **right:** Final result of repeating pattern form Yellow Submarine

```
7, 8, 8, 10              1, 2, 5, 6, 7
1, 2, 5, 4               7, 8, 8, 10, 1, 2
1, 2, 5, 6               7, 8, 8, 9, 7, 8, 8
7, 8, 8, 9               1, 2, 5, 4, 1, 2
7, 8, 8, 10, 1           7, 8, 8, 9, 7, 8, 8, 10
1, 2, 3, 4               1, 2, 3, 4, 1, 2, 3
1, 2, 3, 4, 1            4, 1, 2, 3
1, 2, 5, 4, 1            4, 1, 2, 5
7, 8, 8, 9, 7            8, 8, 10, 1, 2
8, 8, 10, 1              7, 8, 8, 9, 7, 8, 8, 10, 1
2, 5, 4, 1               2, 5, 4, 1, 2, 5
7, 8, 8, 9, 7, 8         6, 7, 8, 8
2, 5, 4, 1, 2            8, 10, 1, 2
1, 2, 3, 4, 1, 2         5, 4, 1, 2
```

Figure 5. Patterns in Modified LZ78 dictionary

4. CONCLUSION AND FUTURE WORK

A dictionary approach was developed in repetitive pattern finding for music feature extraction and indexing. It was shown with experimental results that the proposed method could detect repetitive patterns in music effectively.

In the future, we would like to continue to work on efficient pruning techniques for the proposed dictionary approach to enhance obtained results. Further improvement of the modified Lempel Ziv 78 (MLZ78) will also be carried out to give a better intermediate result for pruning. Also, MIDI files are used as the input to our system so that the bar representation is very clear. However, when a piece of music is either played or sung by people, we have to convert the acoustic waveform to the bar

representation as a preprocessing task. This demands some signal processing techniques. Besides, the bar representation may not be as accurate as that obtained from MIDI files, we have to develop a matching process that permits a certain level of error tolerance.

REFERENCES

[1] A. L. P. Chen, and C. C. Liu, "Music databases: indexing techniques and implementation", *Proceedings IEEE Intl. Workshop on Multimedia Data Base Management Systems*.1999.

[2] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. "Query by humming: musical information retrieval in an audio database", *Proceedings of ACM Multimedia Conference'95*, San Francisco, California, November 1995.

[3] C. C. Liu, J. L. Hsu and A. L. P. Chen, "Efficient theme and non-trivial repeating pattern discovering in music databases," *Proc. IEEE International Conference on Data Engineering.* 1999.

[4] J. L. Hsu, C. C. Liu and A. L. P. Chen, "Efficient repeating pattern finding in music databases," *Proc. ACM Seventh International Conference on Information and Knowledge Management (CIKM).* 1998.

[5] R. J. McNab, *Interactive Applications of Music Transcription,* Master's thesis, Department of Computer Science, University of Waikato, New Zealand, 1996.

[6] R. J. McNab, L. A. Smith, I. H. Witten, C. L. Henderson, and S. J. Cunningham, "Towards the digital music library: Tune retrieval from acoustic input," In *Digital Libraries Conference*, 1996.

[7] T. Zhang and C.-C. J. Kuo, *Content-based Audio Classification and Retrieval for Audiovisual Data Parsing*, Kluwer Academic Publishers, 2001.

[8] J. Ziv and A. Lempel,. "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory,* Volume 23, Number 3, September 1977, pp. 337-343.

[9] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory,* Volume 24, Number 5, September 1978, pp. 530-536.