

Music Indexing with Extracted Main Melody by Using 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

Extraction of repetitive patterns of the main melody in a given music piece is investigated in this research. A dictionary-based approach is proposed to achieve the task. The input to the proposed system is a piece of music consisting of numerical music scores (e.g. the MIDI file format), and other music forms such as the sound wave have to be converted to numerical music scores first. In the system, segmentation is done based on the tempo information and a music score is decomposed into bars. Each bar is indexed, and a bar index table is built accordingly. Then, an adaptive dictionary-based algorithm known as the Lempel Ziv 78 (LZ-78) is modified and applied to the bar-represented music scores to extract repetitive patterns. The LZ78 algorithm is slightly modified to achieve better results, and the modified LZ78 is named the "Exhaustive Search with Progressive Length" (ESPLE). After this step, pruning is applied to this dictionary to remove non-repeating patterns. Modified LZ78 and pruning are repetitively applied to the updated dictionary, which is generated from the previous cycle, until the dictionary converges. Experiments are performed on MIDI files to demonstrate the superior performance of the proposed algorithm.

Keywords: Music database, audio database, music indexing, repeating patterns, Lempel Ziv 78, LZ-78, melody search, pattern search

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. A relatively small amount of effort has been put into audio feature extraction and indexing. Audio database management finds applications in music archiving, special effect sound search for audio editing, etc. Audio is also an integral part of multimedia databases, and often contains useful information for effective multimedia search. Multimedia database management with multi-modal information, such as audio, video and text, is an emerging trend. A better understanding of audio features and their utilization is an essential step towards creating a complete multimedia database management system. In the context of audio databases, music is especially important since it has become a commercial product in our daily life.

Some work has been done in music content analysis and database organization. Chen, *et al.* [1] proposed a pat-tree approach to index melodies, where pat trees were built with chords. The pat tree is a Patricia-like tree, which is a string containing all possible substrings of a given string. Ghias, *et al.* [2] used coarse melody contours as a key to query a music database. McNab, *et al.* [7],[8] used interval contours for interactive music retrieval. Tong and Kuo [9] considered a hidden Markov model (HMM) method to model special effect sounds for content-based audio query. Furthermore, Chen, *et al.* proposed the string-join approach [3] and the correlative matrix approach [4] to find repeating patterns 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 that information to find repeating patterns. Both approaches use notes as the basic units. However, the computational complexity grows rapidly as the number of notes increases. Moreover, the essential duration information of each note was discarded in these systems. In contrast, our proposed system uses bars, instead of notes, as the basic unit. It not only captures the tempo information of melodies but also reduces the size of the input sequence.

In this work, we focus on the extraction of repeating patterns in the main melody of a given music piece. Repeating patterns can be used in organizing and indexing music databases. They also serve as an important feature for content-

based retrieval. Furthermore, they can be used as a tool for analyzing characteristics of compositions and their composers. It is believed that people are particularly sensitive and receptive to certain salient portions in a piece of music. Here, we assume that repeating melodies constitute such a salient part. It is common that a piece of music is composed by certain small pieces of melodies that are repeated throughout the whole piece and can be memorized by people more easily. 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 repeating patterns based on the classic work of Lempel and Ziv [5],[6].

The rest of this paper is organized as follows. In Section 2, some basic concepts and terminology from music theory to be used in this paper are described. The proposed algorithm is described in detail in Section 3. Experimental results are given in Section 4, and concluding remarks are presented in Section 5.

2. BASIC MUSIC THEORY

This section provides a brief review of some basic terms from music theory to be used in this paper.

2.1 Staff

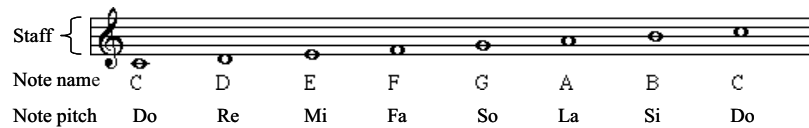


Figure 1. A staff and notes

A staff has five lines and four spaces as shown in Figure 1. Lower lines or spaces of a staff have lower pitches than higher lines or spaces. The lowest line of a staff is center ‘Mi’. If a pitch cannot be drawn within these five lines and four spaces, extra lines can be added. As illustrated in Figure 1, if a center ‘Do’ is out of a staff’s range, an extra line is drawn to represent it.

2.2 Note

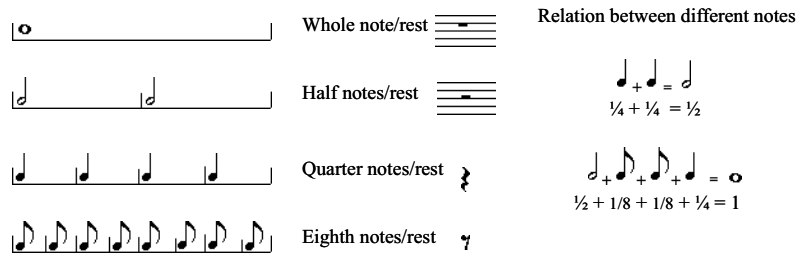


Figure 2. Notes’ symbols, and durations

A note is the basic unit in a music score. It contains pitch and duration information. A rest note is used to denote a rest period in a music score and contains only duration information. A note’s pitch is shown by placing it on a staff. Figure 1 shows the pitch values and the corresponding names of the various notes. The duration of a note or a rest note is represented by a unique symbol. Figure 2 shows the symbols of notes and the relationships between different durations. For example, one whole note equals to two half notes.

2.3 Time Signature

A time signature is used to declare a time unit in a music piece and the number of time units comprising a music segment (referred to as a bar). Usually, a time unit is called a beat. A piece of music may contain more than one time signature. For example, a 3/4 time signature in Figure 3 implies that a quarter (1/4) note is a beat, and three beats form a segment (bar). The second segment of Figure 3 has 4 eighth notes and a quarter note, and the total is 3 beats.



Figure 3. Time signature and bar

2.4 Bar

Music is often divided up into units called bars, as shown in Figure 3. The number of beats in a bar is based on a time signature. Bars under the same time signature have the same duration.

3. PROPOSED ALGORITHM

The input to the proposed system is a piece of music consisting of numerical music scores. Thus, it is assumed that other music forms such as the sound have been first converted to numerical music scores.

3.1 System Overview

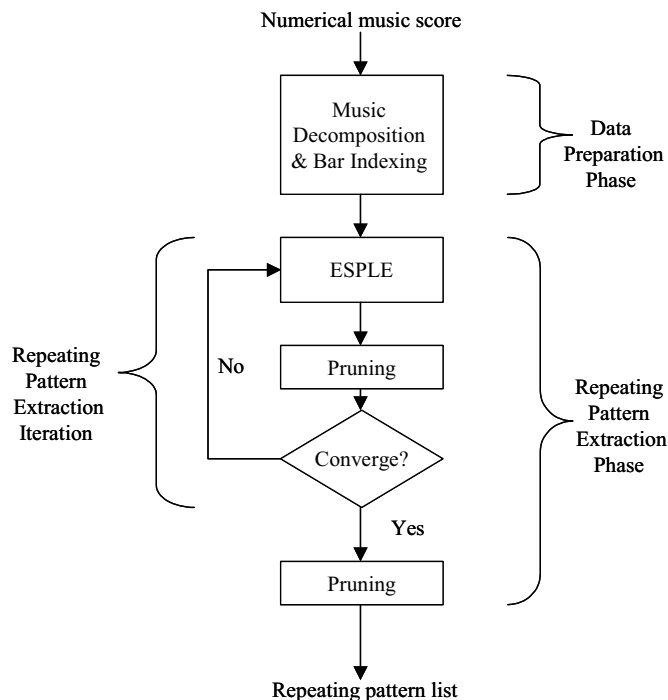


Figure 4. System blockdiagram

Figure 4 is the functional flow-diagram of the overall system. Two similar systems were proposed to solve the same problem in [10] and [11]. In [10], the ESPLE block in Figure 4 was replaced by MLZ78 that was a slightly modified version of the LZ78 process, and the pruning block within the repeating pattern extraction iteration was not introduced.

Thus, a more advance and complicated version of final pruning was required in [10]. To reduce the burden of the final-stage pruning in [10], an immediate pruning step was introduced right after the MLZ78 process in the repeating pattern extraction iteration in [11]. Since the iteration processes are different for all three systems, the convergence criteria are also different. For detailed description of the other two systems, we refer to [10] and [11].

The two main phases in the processing are “data preparation” and “repeating pattern extraction”. Music decomposition and bar indexing constitute the data preparation phase as described below.

1. The numerical music score is first segmented into bars based on the time signature.
2. A bar index table is built according to bars obtained from the segmented numerical music score.
3. Each bar is then replaced by its corresponding index, resulting in the so-called bar indexed music score, which is a term used through out this paper.
4. The bar indexed music score and the bar index table are ready for the next phase of processing.

The two main modules in the repeating pattern extraction phase are: ESPLE processing and dictionary pruning. The extraction of repeating patterns is done iteratively. A repeating pattern list is introduced to store the extracted full-length repeating patterns. A repeating pattern is said to be of full-length if it is not a proper subset of any repeating pattern that has the same frequency count. A dictionary is generated after each ESPLE iteration and pruned to remove non-repeating patterns. Moreover, extracted full-length repeating patterns are moved into a repeating pattern list. The pruned dictionary is passed on to the next ESPLE iteration. The iteration is terminated when the system converges i.e., when the pruned dictionary is empty. Details of the repeating pattern extraction phase are given below.

1. The bar index table and the bar indexed music score generated from the data preparation phase are passed on to the repeating pattern extraction phase.
2. The initial dictionary and the repeating pattern list are empty before any repeating pattern extraction iteration.
3. A new dictionary is generated based on the bar-indexed music score and the pruned dictionary from the previous repeating pattern extraction iteration. (The first iteration utilizes only the initial dictionary).
4. The dictionary is then pruned to remove non-repeating patterns.
5. If a full-length repeating pattern is detected, it is moved from the dictionary to the repeating pattern list.
6. The pruned dictionary of the current iteration is checked for emptiness. If it is indeed empty, go to Step 8. If not, go to Step 7.
7. The pruned dictionary is passed to the next repeating pattern extraction iteration. Go to Step 3.
8. The repeating pattern list is pruned to remove parents’ patterns. Then, the whole process is terminated and the repeating pattern list is returned.

Further details of each of the modules mentioned above are described in the following sections.

3.2 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, a music note is too fine a unit to build a dictionary with since there are too many notes and their combinations in a piece of music, and the complexity of the dictionary building process grows very quickly. Second, as mentioned in Section 2.2, a note contains the pitch and the duration information. However, if a note is used as a symbol to build a dictionary, the duration of a note will be discarded. Therefore, bars are introduced to preserve the duration information of notes. In music scores, there are time signatures used to indicate 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 repeating pattern may not start precisely at the beginning of a bar or stop at the end of a bar. In other words, they may start or stop at any note in a bar. For a given song, repeating 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 the offsets of music notes with respect to the whole piece of music are 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. Just the leading and trailing bars of a repeating pattern require 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 a sequence of non-negative integers, i.e. 0, 1, 2, 3, etc. The bar pattern is the sequence of pitch values of notes that are expressed numerically. The bar index frequency is the number of times a bar appears in a piece of music.

When the bar index table is built, the segmented music score is also concurrently converted into a bar indexed music score. This merely implies replacing each bar in the music score with its corresponding index. 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. For example, let us consider a case where each note is a beat, and each bar has four beats. The two bars “Do-Re-Rest-Mi” and “Do-Re-Mi-Rest”, where “Rest” means a rest note, will be treated as the same by the bar matching process. By making this assumption several different bars that have the same number of notes with the same pitch values will be matched to the same index in the bar index table. However, it should be noted that such combinations of notes occur rarely in the same piece of music. Even if it does happen, the system is designed to looking for repeating patterns. A non-trivial repeating pattern is a sequence of several bars. A single bar appearing in one music piece may quite likely appear in a different piece music. It usually does not contribute toward discrimination since it is quite difficult for people to identify a particular piece after listening to only one bar. Therefore, one bar is too short to be considered as a repeating pattern.

3.3 Lempel-Ziv 78 (LZ78) Algorithm and Its Modification: Exhaustive Search with Progressive Length (ESPLE)

The Lempel-Ziv 78 (LZ78) algorithm 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 while 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, which are also called patterns. The main idea of dictionary-based compression is to detect longer vocabulary entries and encode them with shorter codewords. This process turns out to be a powerful tool in finding repeating patterns in music. The dictionary is the place where repeating patterns are accumulated.

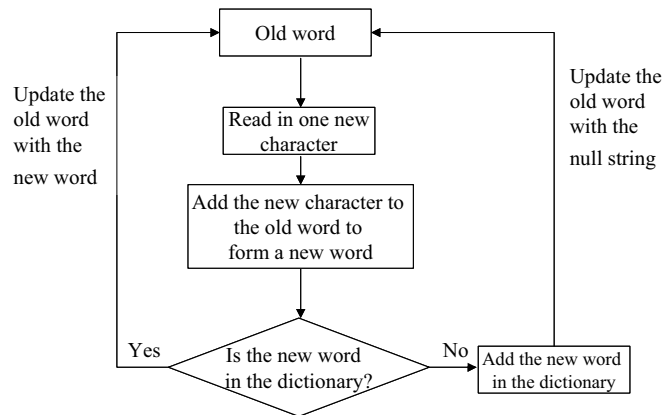


Figure 5. The block diagram of LZ78

The block diagram of LZ78 implementation is shown in Figure 5 and explained below.

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 are stored in the word buffer as shown in Figure 5.
2. One new symbol is read in from input data. In our case, the symbol is a bar index.
3. The new symbol is appended 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, 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, we add this updated word to the dictionary as a new word. Then, the word buffer is emptied. After that, we go back to Step 2.

Following is an example of using LZ78 to process the sequence “DAD_DADABDAD” (note that there are 3 instances of the pattern “DAD”). The updated dictionaries are shown in Figure 6.

1. Symbol “D” is read in and appended to the end of old word buffer. Since initially the old word buffer is empty, the old word becomes new word. Because there is no “D” in the initial dictionary, it is added to the dictionary. The old word buffer is emptied.
2. Next “A” is read in and it has the same situation as the previous “D”. Therefore, “A” is added to the dictionary and the old word buffer is emptied again.
3. “D” is read in, but the dictionary already has a “D” in it. Therefore, “D” remains in the old word buffer, and no update is made to the dictionary.
4. Symbol “_” is read in and appended to the old word, and the old word buffer becomes “D_”. Since the pattern “D_” is not in the dictionary, it is added to the dictionary. The old word buffer is then emptied.
5. “D” is read in, and it is the same situation as in Step 3.
6. “A” is read in, with the same situation as in Step 4. A new word “DA” is added to the dictionary and old word buffer is emptied.
7. “D” read in, and it is the same case as in Step 3.
8. “A” read in, and the old word become “DA”. For the same reason as in Step 3, “DA” remains in the old word buffer, and there is no update to the dictionary.
9. “B” is read in, and the word “DAB” is added to the dictionary and the old word buffer is emptied.
10. After reading in another “D” and “A”, the word “DA” appears again in the old word buffer.
11. The last “D” is read in, and forms “DAD” in the old word buffer. Now “DAD” is added to the dictionary and the old word buffer is emptied.

No	Pattern	No	Pattern	No	Pattern	No	Pattern	No	Pattern	No	Pattern
0	D	0	D	0	D	0	D	0	D	0	D
1		1	A	1	A	1	A	1	A	1	A
2		2		2	D_	2	D_	2	D_	2	D_
3		3		3		3	DA	3	DA	3	DA
4		4		4		4		4	DAB	4	DAB
5		5		5		5		5		5	DAD

Step 1 Step 2 & 3 Step 4 & 5 Step 6, 7 & 8 Step 9 & 10 Step 11

Figure 6. Dictionary of LZ78 for example “DAD_DADABDAD“ from Step 1 to Step 11

From the example given above, we see that identifying the 3-symbol repeating pattern “DAD” requires that the string “DAD” appears at least three times in the input sequence. Also note that the non-repeating pattern “DAB” which appears only once is also caught in the dictionary. “DA” is called the parent pattern of both “DAD” and “DAB” or, equivalently, “DAD” and “DAB” are the children of “DA”. For patterns “DAD” or “DAB” to be in the dictionary, the parent pattern “DA” needs to appear at least twice. In general, to form a parent pattern that is N bars long by using LZ78 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 parent pattern. A long parent pattern is needed for longer repeating patterns. To overcome this difficulty, we pass the same music piece through the LZ78 dictionary building system several times, and the dictionary in each LZ78 iteration is built based on the previously built dictionary. According to our experiments, it is sufficient to have 3 or 4 iterations to extract a repeating pattern that is about 12 bars long.

It is clear from the above example that not every pattern in the dictionary is a repeating pattern. Furthermore, repeated iterations of LZ78 will not necessarily yield a desired dictionary which will converge in a limited number of iterations.

The dictionary will continue to grow with repeated LZ78 iterations (eventually bound only by the entry of the whole music sequence as a dictionary word), and the system tends towards divergence. The system is said to converge if the dictionary of current iteration is the same as the dictionary of the previous iteration. Pruning becomes essential to assure a converging dictionary.

Before detailed discussion of the pruning process, let us summarize three issues related to dictionary building. First, many bars that are not repeating in the bar index table may be included in extracted patterns, since the LZ78 process is applied to the same music more than once. The second issue relates to determining the end of a repeating pattern. It is difficult to find the end point in the pruning phase by using this dictionary. Third, the starting note of a potential repeating pattern may not be aligned well with the extracted patterns in the bar. For example a repeating pattern "ABCDE", "BCDE" may be caught in the dictionary instead of "ABCDE". To address these issues, LZ78 has to be modified to get a better dictionary.

Before describing the ESPLE algorithm, the format of a dictionary should be explained. A dictionary contains 5 attributes: (1) the dictionary index number, (2) patterns, (3) the frequency, (4) the pattern length, and (5) positions of the pattern. The dictionary index number is a unique number for each pattern in the dictionary. A pattern of combination of several bars is recorded in the pattern attribute. The frequency is used to record the number of times a pattern appears. The pattern length is used to keep track of the length of pattern. Positions of a pattern are used to keep track of the pattern's appearance in an input string.

First, instead of applying LZ78 once in an iteration for an input string, a pattern extension is applied to multiple substrings which are generated from the input string. If an input string is ordered from the left to the right, multiple substrings are generated by shifting the starting point to the right for each position in the input string. For example, if the input string is "ABCDE", then the multiple input substrings are "ABCDE", "BCDE", "CDE", "DE", and "E". Each iteration only extends the length of the patterns by one. For the same example used previously, "A", "B", "C", "D", and "E" are the patterns after the first iteration. "AB", "BC", "CD", and "DE" are the newly generated patterns after the second iteration. If the starting point of a substring covered by the previously extended pattern is eliminated, then the iteration is similar to LZ78 iteration.

Second, since the frequency of each index is recorded in the bar index table when it is first built (see Sec. 3.2) it is available prior to the execution of the ESPLE algorithm. This bar index frequency information can be used to improve the effectiveness of ESPLE. If the bar index frequency is one, clearly the corresponding index number should not be a part of any repeating pattern. Hence, such non-repeating bars should be ignored from being included in any of the extended patterns. Therefore, any pattern that is going to include a non-repeating bar should be discarded. In other words, in the pattern expanding process, a pattern in the old word buffer should stop expanding into longer patterns, when a non-repeating bar is read in. This modification guides the expanding process to stop expanding a repeating pattern into another non-repeating pattern. This procedure also helps in successfully determining end points of some of repeating patterns. To determine end points of the remaining repeating patterns, yet another technique is used during dictionary pruning. To enable this method, a "pattern frequency" attribute is included in the dictionary. It is used to count the number of times each pattern appears. The pattern frequency will be updated in the pruning phase. Non-repeating patterns will be removed from the dictionary. The third issue (as mentioned earlier) will be solved at the pruning phase.

The flow diagram of ESPLE is shown in Figure 7, and described below.

1. A buffer is needed in ESPLE for each substring, and only one dictionary is required for all ESPLE iterations. Initially buffers for different substrings and the dictionary are empty. The buffer is referred to as old/new word.
2. One new character is read in from the incoming substring. If the bar index frequency is 1, terminate ESPLE iteration for the current substring. There is no more ESPLE iteration for this substring in the future iteration.
3. Append the new character to the old word, and it becomes the new word. There is at least one character in the new word buffer.
4. There are two cases. (a) If the new word is already in the dictionary, then this new word becomes the old word and nothing is changed in the buffer. Update the corresponding word's frequency and position in the dictionary. Save the old word buffer for the next iteration. (b) If the new word is not in the dictionary, add the new word to

the dictionary. Then, record the pattern length and appearance position of the newly added pattern. Save the old word buffer for the next iteration.

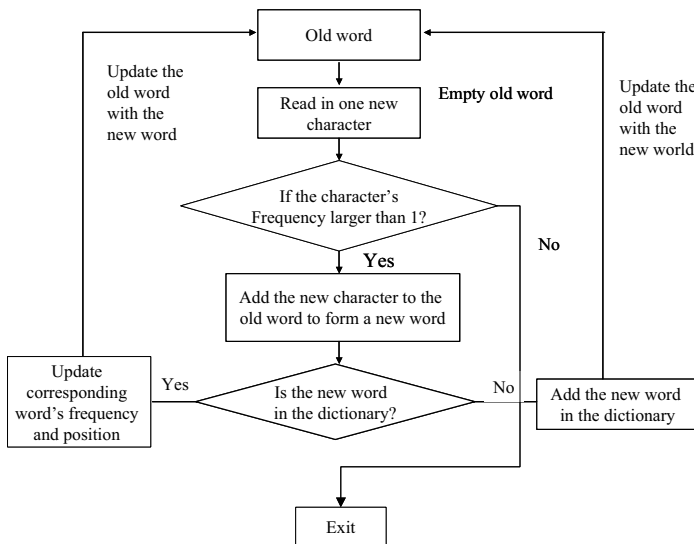


Figure 7. The block diagram of ESPLE

Let us consider the same example used earlier to illustrate LZ78. Let “DAD_DADABDAD” be the input sequence, and its bar index table and updated (intermediate) dictionaries are shown in Figure 8.

1. “D” is read in for substring “DAD_DADABDAD” and is appended to the end of the old word buffer. Since initially the old word buffer is empty, the old word becomes the new word. Because there is no “D” in the initial dictionary, a “D” is added to the dictionary. The old word buffer of the current input substring is kept for the next ESPLE iteration.
2. “A” is read in for substring “AD_DADABDAD” and it has the same situation as “D” in the previous step. Therefore, “A” is added to the dictionary and the old word buffer of the current substring is kept for the next ESPLE iteration.
3. “D” is read in for substring “D_DADABDAD”, but the dictionary already has a “D” in it. Therefore, the frequency of “D” is updated and the old word buffer of the current substring is kept for the next iteration.
4. “_” is read in for substring “_DADABDAD”, but since it appears only once in the input sequence, its bar index frequency is 1. The iteration of the current substring, which is “_DADABDAD”, is terminated and no further expanding is performed in later iterations. Note that “_” is not added into the dictionary.
5. Apply ESPLE to each substring. If the substring is shorter than the required pattern length, it is discarded.
6. For the 2nd iteration, 2 more vocabularies with length 2 are generated in the dictionary.
7. For the 3rd iteration, additional 2 vocabularies with length 3 are added to the dictionary.

No	Pattern
0	D
1	A
2	
3	
4	
5	

Step 1 ~ 5

No	Pattern
0	D
1	A
2	DA
3	AD
4	
5	

Step 6

No	Pattern
0	D
1	A
2	DA
3	AD
4	DAD
5	ADA

Step 7

Index	Frequency
A	4
B	1
D	6
_	1

Bar index table

Figure 8. Bar index table and dictionaries for example of “DAD_DADABDAD” with ESPLE

A comparison of the dictionaries of LZ78 (Figure 6) and ESPLE (Figure 8) for the example shows that ESPLE results in a similar dictionary size. Patterns with non-repeating bar indices are not seen in the dictionary shown in Figure 8.

However, this specific example shows only three iterations of ESPLE. Moreover, the underlying repeating pattern is very short. In general, a dictionary after several ESPLE iterations may not be as simple as given in this example. Patterns in the dictionary may not be repeating patterns. Patterns are overlapping since substrings are overlapping. Furthermore all parents of patterns are also included in the dictionary. The dictionary will diverge if these non-repeating patterns, overlapping patterns and pattern's parents are not handled properly. Hence, pruning the dictionary after each ESPLE iteration is essential to have a convergent dictionary, thus enabling easier extraction of repeating patterns. Details of the pruning techniques are discussed in the next section.

3.4 Pruning

Figure 9 shows the flowdiagram of the pruning algorithm. The pruning phase has three stages: repeating pattern verification, repeating pattern extraction, and pattern elimination. In the first stage (i.e. repeating pattern verification), the repetition of each pattern is verified by using the frequency attribute in the dictionary. As mentioned in the previous section, non-repeating patterns will also be caught in the dictionary, so verifying repetitions becomes necessary. Although the proposed ESPLE algorithm tries to eliminate the problem of having non-repeating patterns in the dictionary, some non-repeating patterns may still appear in the dictionary. The main reason for this phenomenon is a bar index that is not a part of any repeating patterns may have multiple appearances in a piece of music. These appearances typically occur in isolated places and the ESPLE algorithm has no way of detecting this problem. Since the dictionary converges to empty, the time required for checking repetitions will not grow as the number of ESPLE iterations increases. All patterns in the dictionary will be checked for their repetition, while at the same time the frequency attribute in the dictionary will be updated.

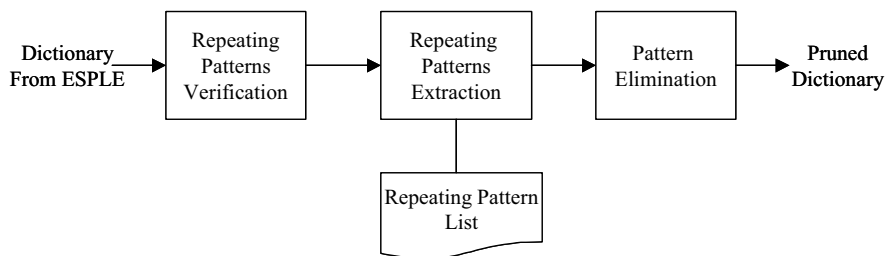


Figure 9. The block diagram of the pruning phase

In the repeating pattern extraction stage, full-length repeating patterns are extracted and moved from the dictionary to a repeating pattern list. That is, entries of detected full-length repeating patterns are no longer retained in the dictionary. As mentioned in Section 3.1, a repeating pattern is said to be a full-length repeating pattern, if it is not a proper subset of any repeating pattern which has the same frequency. A subset of a full-length repeating pattern may have a higher frequency than the full-length repeating pattern. Then, this subset may be another full-length repeating pattern if it is not a proper subset of other repeating patterns that have the same frequencies. All entries in the repeating pattern list are full-length repeating patterns. A threshold will be set to tell the system what the minimum length of a pattern should be for it to be considered as a repeating pattern. In practice, the threshold cannot be set too short or too long, since a too short pattern cannot be considered as a valid repeating pattern, and a too long pattern may cause no repeating patterns to be found. There is a way of determining repeating patterns by detecting non-repeating patterns. A pattern of length N is not a repeating pattern but its parent patterns of length N-1 must be repeating patterns. Otherwise, the non-repeating patterns will not be able to extend to length N. As mentioned in the previous section, the fourth attribute of a dictionary is used to keep track of a pattern's length. That is, the dictionary keeps track of each pattern's parent. Then, extracting a full-length repeating pattern from the dictionary becomes straightforward.

In the pattern elimination stage, non-repeating patterns are removed from the dictionary. Patterns which are overlapping are removed. For example, in the string "ABCABCABCDEFABC", the pattern "ABCABC" will be detected twice which has a middle "ABC" overlapping. To solve this problem, the attributes of the pattern's position and pattern's length are used. As a result, we now have "ABC" in our current dictionary. The positions of appearance of "ABC" are 0,3,6,12. The method to detect the overlapping is easy. Since the length of pattern is known (3 in the current example),

the positions are checked from the first appearance to the last. The first position is 0, and no previous pattern has reached this position yet. Then, the second position is 3 which is reached by the first pattern appearance (current position + pattern length) i.e. $(0+3)=3$. Then, the first position, which is 0, is removed. Since the second appearance makes the first appearance disappear, the second appearance, which is position 3, should be retained. We skip to the next position that is 12. However, since for the possible third position $(6 + 3)=9$, the third appearance will not reach 12 and position 12 is kept. This solves the overlapping problem.

Finally, after the pruned dictionary becomes empty, the repeating pattern list needs to be pruned. Patterns of length less than a preset minimum pattern length are removed. Moreover, proper subsets of an extracted full-length repeating pattern will be removed as well.

4. EXPERIMENTAL RESULTS

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 VC++ 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 use the piece "Yellow Submarine" by Beatles as an example. Figure 10 shows the first nine bars of Yellow Submarine in music scores and the numerical pitch values are specified under each note (the pitch values are obtained from the MIDI file). In a MIDI file, the range of pitch is between 0 to 255, and hence ASCII characters are used to replace the numerical pitch in our experiment. After converting the music score to the numerical music score, it 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 based on the bar index table. Table 1 shows the entire bar index table for Yellow Submarine.

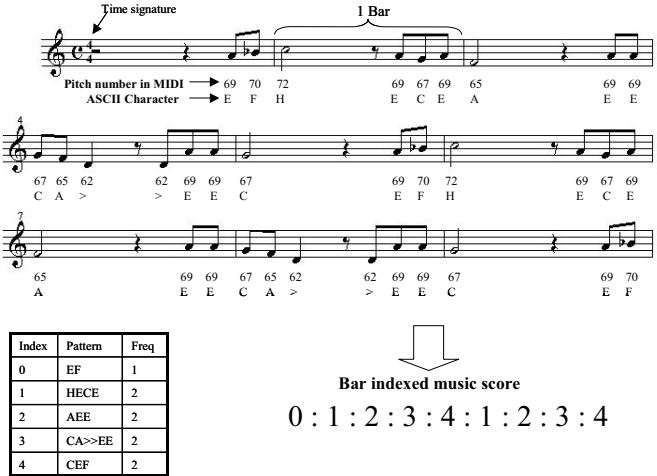


Figure 10. The First nine bars of the music score of “Yellow Submarine”

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

Figure 11. The bar index music score (left) and the final result of repeating patterns (right) in Yellow Submarine

The left part of Figure 11 shows the entire bar indexed music score. We applied ESPLE to the bar indexed music score and pruned the dictionary over several iterations until the dictionary finally converged. The dictionary of this example

converged after 15 iterations. The threshold for the minimum length of a repeating pattern was set to 3. Finally, the algorithm extracted 5 repeating patterns, which are shown in the right part of Figure 11.

Index	Pattern	Freq.	Index	Pattern	Freq.
0	EF	1	6	C	2
1	HECE	10	7	HHHJ	6
2	AEE	9	8	CCCC	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 number of extracted repeating patterns was found to vary across different songs. Some songs tended to have more repeating patterns than others. The length of repeating patterns also varied across songs. The number of segmented bars of a piece of music is dependent on the length of the input music. However, the bar index table of a piece of music will be dependent on the nature of the piece of music. In our system, we only set the minimum length threshold since very short repeating patterns were not deemed to be important. Some non-trivial (i.e., lengthy) repeating patterns may be extracted from a music piece. In fact, a repeating pattern in a full-length repeating pattern may or may not sound as a complete repeating melody, since only certain combinations of notes can be used to end a melody. Therefore, the proposed system is used only to extract full-length repeating patterns. Table 2 shows the statistical information of the tested songs. As seen in Table 2, some repeating patterns end up being very long after pruning. For example, Hotel California by Eagle has only two very long repeating patterns that are truly the two main melodies as confirmed by informal listening tests.

5. CONCLUSION AND FUTURE WORK

Title	NO. of bars	Bar index table Size	NO. of ESPLE iterations	NO. of repeating patterns	Max. repeating pattern Length
Yellow Submarine	59	12	18	5	18
All I have to do is dream	55	27	5	3	5
500 miles	52	24	6	5	6
Hotel California	117	22	15	2	15

Table 2. Statistical information of tested songs

A dictionary-based approach was developed to find repeating patterns for music feature extraction and indexing. It was shown with experimental results that the proposed method could detect repeating patterns in music effectively.

In the future, we would like to continue our work on efficient pruning techniques for the proposed dictionary approach to enhance obtained results. Further improvement of ESPLE will also be carried out to give a better intermediate result for pruning. Also, since MIDI files are used as the input to our system, the bar representation used for pattern extraction is unambiguous. However, when a piece of music is either played or sung by people, we have to convert the acoustic waveform to the bar representation in a preprocessing step. This demands robust signal processing techniques. Besides,

since 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. This may require statistical approaches to music pattern extraction.

6. 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] 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.
- [6] 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.
- [7] R. J. McNab, *Interactive Applications of Music Transcription*, Master's thesis, Department of Computer Science, University of Waikato, New Zealand, 1996.
- [8] 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.
- [9] T. Zhang and C.-C. J. Kuo, *Content-based Audio Classification and Retrieval for Audiovisual Data Parsing*, Kluwer Academic Publishers, 2001.
- [10] H.-H. Shih, S. S. Narayanan and C.-C. J. Kuo, "Automatic Main Melody Extraction From Midi Files With A Modified Lempel-ZIV Algorithm," *2001 International Symposium on Intelligent Multimedia, Video & Speech Processing*, May 2001.
- [11] H.-H. Shih, S. S. Narayanan and C.-C. J. Kuo, "A Dictionary Approach to Repetitive Pattern Finding in Music," *2001 IEEE International Conference on Multimedia and Expo (ICME2001)*, August 2001.