# Finding Approximate Repeating Patterns from Sequence Data

Jia-Lien Hsu
Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taiwan, R.O.C.
E-mail: alien@csie.fju.edu.tw

Arbee L.P. Chen
Department of Computer Science National Chengchi University Taipei, Taiwan, R.O.C.
E-mail: alpchen@cs.nccu.edu.tw

Hung-Chen Chen
Department of Computer Science National Tsing Hua University Hsinchu, Taiwan, R.O.C.
E-mail: jesse@cs.nthu.edu.tw

## ABSTRACT

In this paper, an application of feature extraction from music data is first introduced to motivate our research of finding approximate repeating patterns from sequence data. An approximate repeating pattern is defined as a sequence of symbols which appears more than once under certain approximation types in a data sequence. By using the '*cut*' and '*pattern_join*' operators, we develop a level-wise approach to solve the problem of finding approximate repeating patterns.

## 1. INTRODUCTION

Compared to transactional data, less attention on data mining has been drawn to the issues of mining sequence data such as traces of web browsing activities and sequences of multimedia data. Although tasks of data mining are usually application-dependent, to discover some universal properties, such as repetitions and trends, from data sequences is still promising.

### 1.1 Application: Feature Extraction from Music Data

For content-based music data retrieval, one of fundamental techniques is to extract music features from the raw data of music objects and organize them as a music index for further processing.

Taking into account of the music characteristics, the music features can be classified into four categories: *static music information*, *acoustical feature*, *thematic feature* and *structural feature* [Hsu01]. As for the structural feature, classic music objects are composed according to a special structure called *musical form* in which there are two basic rules: hierarchical rule and repetition rule [Jone74][Krum90][Narm90]. The hierarchical rule says music objects are formed hierarchically. The repetition rule says that some sequences of notes, known as *motives*, repeatedly appear in a movement. Repetition rule is also meaningful for other music categories. For example, the repetition in pop music is called the *refrain*.

Based on the repetition rule, we derive the sequences of notes appearing more than once in the music object as its structural feature. The sequences are called *repeating patterns* [Hsu01][Hsu98]. Researchers in the musicology field also agree that repetition is a universal characteristic in music structure modeling [Krum90][Narm90]. Meanwhile, the length of repeating patterns is much shorter than that of a music object. Choosing repeating patterns as the features to represent the music objects meets both efficiency and semantic-richness requirements for content-based music data retrieval. Therefore, techniques for finding the repeating patterns from the sequence of notes of a music object need are to be developed.

However, patterns may repeat in the music object with some variance. One of the concepts to deal with this variance is the *prototypical melody*. "The prototypical melody is a kind of generalization to which elements of information represented in the actual melody may seem relevant" [Self98]. The prototypical melody suggests the greatest influence on the way the actual melody is remembered and retrieved. For example, consider the five extracts from Mozart's Piano Sonata K.311, shown in Figure 1(a)-(e). A prototypical melody, which approximates the five extracts, is identified in Figure 1(f).

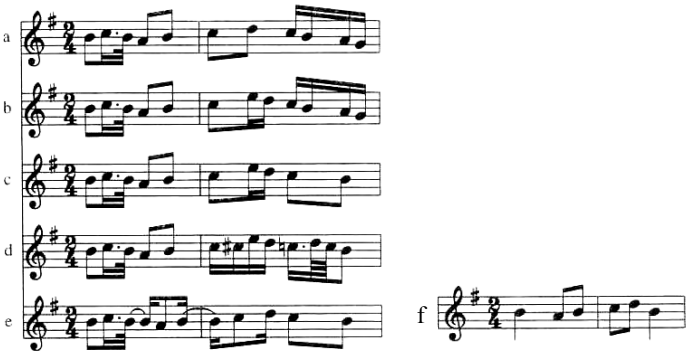


**Figure 1: Five extracts from Mozart's Piano Sonata K. 311 and a prototypical melody (excerpted from [Self98]).**

For the purpose of searching, it is easier to handle the compositions which are managed in a consistent way to extract features. An algorithmic approach to the problem of identifying prototypical melody is required for music information retrieval.

### 1.2 Related Works

Considering the previous application on music data, Shih, *et al.* [Shih01] propose a modified Lempel-Ziv algorithm for automatic extraction of exact repeating patterns in music databases. The music objects are first segmented into *bars* and the bar index table is then constructed. An adaptive dictionary-based compression algorithm (LZ-78) is then applied to the bar-represented music scores to extract repetitive patterns. Rolland [Roll98][Roll99] also focus on the pattern extraction problem and propose a more flexible similarity metrics between music sequences. A dynamic programming-based approach, called FlExPat, is also introduced. By pair comparison and then categorization, the melodic patterns can be found.

In [Meek01], the authors introduce an algorithm, Melodic Motive Extractor (MME), to "extract themes from a piece of music." Based on the hashing function techniques and lattice structure, the MME is devised to identify frequent patterns from a sequence of music contour. In [Dann02], the authors apply the dynamic programming technique on music audio data, which is represented as sequences, to recognize the repetition structure. First, possible pairs of *segment* (*i.e.*, subsequence) will be identified and considered as candidates. Similar candidates will be clustered, and the analysis of musical structure will also be produced according. However, the time complexity of proposed method would be as higher as $O(n^4)$, where $n$ is the length of sequence. In [Pien02], a text-based method is applied to extract maximal frequent phrase from music data. The proposed approach is a variant of $n$-gram method by combining bottom-up and

greedy methods, and first introduced in [Ahon99] for text mining task from documents.

We propose two approaches in discovering repeating patterns in music data [Hsu01][Hsu98]. For the first approach, the repeating patterns are found based on a data structure called *correlative matrix*. For a music object of $n$ notes, an ($n$ x $n$) correlative matrix is constructed to keep the intermediate results during the finding process. In the other approach, the longer repeating pattern of a music object is discovered by repeatedly combining shorter repeating patterns by a *string-join* operation. The storage space and execution time can therefore be reduced. In this paper, we extend the problem of finding exact repeating patterns to finding approximate repeating patterns.

## 2. PROBLEM FORMULATION

The application in Section 1 motivate the problem of finding approximate repeating patterns from sequence data. In this section, we formulate the problem and introduce three types of approximations, *i.e.*, *longer_length*, *shorter_length*, and *equal_length*. Due to the space limitation, we only discuss the case of longer_length approximation in the rest of this paper.

### 2.1 The Definitions

We first define the match operator, *i.e.*, *longer_length_match*, as follows.

DEFINITION 2.1: *longer_length_match* (P, LL)

Given P = ($p_1$, $p_2$, … , $p_m$) which is a pattern sequence of length $m$, and LL = ($s_1$, $s_2$, … , $s_n$), a data sequence of length $n$, where $n > m$.

$$longer\_length\_match(\mathrm{P},\mathrm{LL}) = \begin{cases} 1, & \text{if } p_i = s_{b_i}, \text{ for } i = 1,2,\cdots,m, \\ & \text{where } 1 = b_1 < b_2 < \cdots < b_m = n \\ 0, & \text{otherwise.} \end{cases}$$

Define $r = n - m$ to indicate the *approximation degree* of *longer_length_match* (P, LL). *longer_length_match* (P, LL) = 1 when there exist $m$ symbols in LL, which match the $m$ symbols in P in sequence, and $s_1 = p_1$, $s_n = p_m$. The approximation degree denotes the number of symbols which will not be matched when applying a match operator.

For example, let P be a four-symbol pattern, P = ($p_1$, $p_2$, $p_3$, $p_4$) = (A, B, C, D). Let LL be a six-symbol pattern, LL = ($s_1$, $s_2$, … , $s_6$) = (A, B, K, C, M, D). In this case, we can find ($b_1$, $b_2$, $b_3$, $b_4$) = (1, 2, 4, 6), which means that $p_1$ matches $s_1$, $p_2$ matches $s_2$, $p_3$ matches $s_4$, and $p_4$ matches $s_6$. Therefore, *longer_length_match* (P, LL) = 1, and the approximation degree is two.

The match operator is used to compute the *repeating frequency* of a pattern sequence P in a data sequence S. The repeating frequency of P is the number of appearance of P in data sequence S. Each appearance is identified by a substring of S which makes the match operator satisfied. We discuss the computation of the repeating frequency of P in S with approximation degree $r$ as follows.

Denote *freq* (P, S, $r$, $AT$) as the repeating frequency of pattern sequence P in a data sequence S with the approximation type $AT$ (*i.e.*, $AT$ = longer_length) and approximation degree $r$.

- $$freq(\mathrm{P},\mathrm{S},r,\text{longer\_length}) = \sum longer\_length\_match(\mathrm{P},\mathrm{LL}_i)$$

  where
  (1) $\mathrm{LL}_i$ is a substring of S
  (2) $|\mathrm{LL}_i| = |\mathrm{P}| + r$
  (3) for any $\mathrm{LL}_i = \mathrm{S}[a..b]$ and $\mathrm{LL}_j = \mathrm{S}[c..d]$, $i \neq j$, if $longer\_length\_match(\mathrm{P},\mathrm{LL}_i) = 1$ and $longer\_length\_match(\mathrm{P},\mathrm{LL}_j) = 1$, then either $b < c$ or $d < a$ must hold.

Each appearance is identified by a substring of S which satisfies the *longer_length_match* operator. Moreover, there is no overlap among these appearances as specified in (3).

As discussed in Section 1, the found patterns can be refrains and motives of music objects. The refrains and motives are recognizable patterns which repeat several times. Therefore, the overlapped appearances cannot be considered as recognizable patterns.

**Example 1**

Let P be "ABC", and S be "AKBCDEABLCF". Consider the longer_length approximation with degree one. Among all substrings of length four, we have two substrings, $\mathrm{LL}_1$ and $\mathrm{LL}_2$, which set *longer_length_match* (P, $\mathrm{LL}_i$) to 1, *i.e.*, "AKBC" and "ABLC". Therefore, *freq* ("ABC", "AKBCDEABLCF", 1, longer_length) = 2.

### 2.2 The Problem

The problem of finding approximate repeating patterns is formulated as follows. Given a data sequence S, and the parameters of pattern length, approximation degree, minimal repeating frequency, and approximation type (denoted *pa_i*, *pa_r*, *pa_f*, and *AT* respectively), find all approximate repeating patterns. The *pa_i* specifies the range of pattern length to be found. The *pa_r* specifies the range of approximation degree, specifically, *pa_r* = {0, 1, …, *max_pa_r*} . The *pa_f* specifies the minimal number of pattern appearances to form a repeating pattern.

With respect to the longer_length approximation type, the problem is to find those patterns that repeatedly appear in S, in which the appearances are identified by the *longer_length_match* operator and the repeating frequencies are computed by *freq* (P, S, *r*, longer_length). Therefore, the problem of extracting prototypical melody from music data, as shown in Figure 1, can be formulated as the one of find approximate repeating pattern with *AT* = longer_length. Note that the patterns found are not necessarily substrings of the data sequence. Otherwise, the prototypical melody will not be discovered in any way.

**Example 2**

The data sequence S is "ABFCDLBMABPFCFD", and the parameters are *pa_i* = {1, 2, 3, 4}, *pa_r* = {0, 1}, *pa_f* = 2, and *AT* = longer_length. The setting of parameters means that we are interested in those patterns of length one to four with the approximation type of longer_length. For each appearance of a pattern, at most one symbol of the appearance is not matched when applying *longer_length_match* operator. Each of the found patterns has to appear at least twice in the data sequence. The found patterns are as follows.

$\mathrm{P}_1$ = {"A", "B", "C", "D", "F"},
$\mathrm{P}_2$ = {"AB","BF","CD","FC","FD"},
$\mathrm{P}_3$ = {"ABF","BFC","FCD"}, and $\mathrm{P}_4$ = {"ABFC"}

As an example of the pattern "ABF", since the parameter *pa_r* is set to {0, 1}, we have *freq* ("ABF", S, 0, longer_length) = 1 and *freq* ("ABF", S, 1, longer_length) = 1. In total, there are two appearances of the pattern "ABF", which satisfies the parameter *pa_f*.

## 3. OUR APPROACH

In this section, we propose our solutions to the problem of finding approximate repeating patterns, as well as the concept of *cut* and *pattern_join* operator, denoted by *PJ*.

### 3.1 The Level-wise Approach

To find all approximate repeating patterns, intuitively, we can apply sliding windows of all possible lengths, ranging from one to (*max_pa_i* + *max_pa_r*), over the data sequence S to have a set of substrings. For these substrings, we check whether an approximate repeating pattern can be formed by the *longer_length_match* operator. As in Example 2, we need sliding windows of lengths one to six. Such brute-force process has too many substrings for the checking. In the following, we introduce the concept of cut. By carefully dividing S, we can have fewer substrings for the checking.

Denote *max_pa_i* and *max_pa_r* as the maximal values in the range of *pa_i* and *pa_r*, respectively, and *strlen* (S) as the length of S.

$$cut_i = \mathrm{S}[a\,..\,b], i = 1, 2, 3, \cdots$$
$$\text{where} \quad a = 1 + (cw \times (i-1)), cw = max\_pa\_i + max\_pa\_r$$
$$b = min((2 \times cw - 1) + (cw \times (i-1)), strlen(\mathrm{S})),$$

and $i$ is $cut\_id$.

**Example 3**

As in Example 2, the data sequence S = "ABFCDLBMABPFCFD", *max_pa_i* = 4, and *max_pa_r* = 1. Accordingly, *cw* = *max_pa_i* + *max_pa_r* = 5, we have three cuts as follows.

$cut_1$ = "ABFCDLBMA"
$cut_2$ = "LBMABPFCF"
$cut_3$ = "PFCFD"

Since the length of patterns to be found is bounded by the parameters *pa_i* and *pa_r*, we first partition the sequence S into substrings of length *cw*, the summation of the maximal values of *pa_i* and *pa_r*. However, some patterns may span over two adjacent substrings, therefore we add a padding of length (*cw*−1) for each substring. Note that in (2), the *min* function is used for a boundary condition, in case the last cut has fewer than (2×*cw*−1) symbols.

Before providing the definition of pattern_join operator, we introduce a data structure to represent the found patterns and to keep the information needed for processing. The pattern set of length *i*, denoted by $P_i$ = {<$pat_i^{(1)}$, $plist_i^{(1)}$>, <$pat_i^{(2)}$, $plist_i^{(2)}$>, …, <$pat_i^{(j)}$, $plist_i^{(j)}$>}, where $pat_i^{(j)}$ denotes the *j*-th pattern in $P_i$, and $plist_i^{(j)}$ is a list of triplets (cut_id: start, end). Each triplet indicates an appearance of $pat_i^{(j)}$ in S. The 'cut_id' indicates a cut, and the 'start and 'end' indicate where the pattern $pat_i^{(j)}$ is located in the cut. For example, $P_2$ = {<"BF", (1: 2, 3), (2: 5, 7)>, <"FD", (1: 3, 5), (3: 4, 5)>}. The $P_2$ means that we have two patterns, "BF" and "FD". For the triplet associated to "BF", (2: 5, 7) means that "BF" is located in the $cut_2$ ranging from the fifth to the seventh position.

The triplet whose 'start' value is larger than *cw* is called a *dummy triplet*. The dummy triplets are used for concatenating in succeeding processing. An appearance of a pattern specified by a dummy triplet will also be specified by a non-dummy triplet. Therefore, the repeating frequency of $pat_i^{(j)}$ is the number of triplets, excluding dummy triplets, in $plist_i^{(j)}$. For example, $P_2$ = {<"FC", (1: 3, 4), (2: 7, 8), (3: 2, 3)>}, and the repeating frequency of "FC" is 2.

In the following, we introduce the pattern_join operator. The pattern_join operator is used for concatenating the found patterns of length *i* to derive the candidate patterns of length *i*+1. By applying the pattern_join operator in a level-wise manner, all the patterns will be found.

For two patterns of length *i*, $pat_i^{(a)}$ and $pat_i^{(b)}$, we define the pattern_join operator as follows.

DEFINITION 3.1: pattern_join operator

$$PJ(\langle pat_i^{(a)}, plist_i^{(a)}\rangle, \langle pat_i^{(b)}, plist_i^{(b)}\rangle)$$
$$= \begin{cases} \langle pat_{i+1}^{(c)}, plist_{i+1}^{(c)}\rangle, & \text{if } pat_i^{(a)}[2..i] = pat_i^{(b)}[1..(i\text{-}1)] \\ \varnothing, & \text{otherwise} \end{cases}$$

where

(1) $pat_{i+1}^{(c)}$ = $pat_i^{(a)}[1..i]$ + $pat_i^{(b)}[i..i]$, where the '+' denotes the string concatenation
(2) for (cut_id$^{(a)}$: start$^{(a)}$, end$^{(a)}$) and (cut_id$^{(b)}$: start$^{(b)}$, end$^{(b)}$) from $plist_i^{(a)}$ and $plist_i^{(b)}$, respectively, if
   i. cut_id$^{(a)}$ = cut_id$^{(b)}$ and start$^{(a)}$ < start$^{(b)}$
   ii. $0 \le (\text{end}^{(b)} - \text{start}^{(a)} + 1) - |pat_{i+1}^{(c)}| \le max\_pa\_r$
   add (cut_id$^{(a)}$: start$^{(a)}$, end$^{(b)}$) into $plist_{i+1}^{(c)}$

For two patterns of length *i*, $pat_i^{(a)}$ and $pat_i^{(b)}$, if the two patterns have an overlapping of (*i*–1) symbols, we concatenate the two patterns as the pattern $pat_{i+1}^{(c)}$. Then, we check the corresponding triplet lists to derive the triplet list of $pat_{i+1}^{(c)}$. The triplet list of $pat_{i+1}^{(c)}$ is constructed as follows. The conditions of (2) are used to make sure that the pattern $pat_{i+1}^{(c)}$ and the substring, indicated by the triplet (cut_id$^{(a)}$: start$^{(a)}$, end$^{(b)}$), satisfy the *longer_length_match* operator.

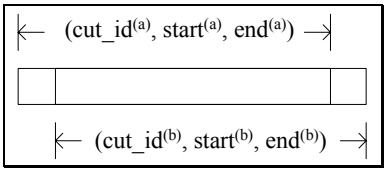


**Figure 2: An illustration of the pattern_join operator.**

**Example 4**

*PJ* (<"BF", (1: 2, 3), (2: 5, 7)>, <"FD", (1: 3, 5), (3: 4, 5)>) = <"BFD", (1: 2, 5)>

Our method is a level-wise approach (procedure `find_approxi_pattern`). First, we determine cuts from the data sequence S. By concatenating the patterns of length *i* from $P_i$ (procedure `find_level`), we derive the candidate patterns of length (*i*+1), denoted by $C_{i+1}$. After checking the repeating frequency of candidate patterns, the patterns of length (*i*+1), $P_{i+1}$, will be confirmed (procedure `prune`). As for the next level, similar processing is performed until all patterns are found.

The main parts of our algorithm are shown as follows. Due to the space limitation, other supporting procedures of our approach are not included in this paper.

```
Algorithm find_approxi_pattern (S, pa_i, pa_r, pa_f)
//input: the data sequence S, pa_i, pa_r, pa_f
//output: the approximate pattern set AP
```

```
Begin
1.   W = cut_dataseq(S, max_pa_i + max_pa_r)
2.   C1 = find_level_1(W)
3.   P1 = prune(C1)
4.   for (i = 1 to (max_pa_i - 1))
5.      Ci+1 = find_level(Pi)
6.      Pi+1 = prune(Ci+1)
7.   AP = P1 ∪ P2 ∪ … ∪ Pmax_pa_i
8.   return AP
End
```

```
Algorithm find_level (Pi)
//input: the pattern set, Pi
//output: the pattern set, Pi+1
Begin
1.   Pi+1 = ∅
2.   for each (A, B) in Pi
3.      TP = pattern_join(A, B)
4.      Pi+1 = Pi+1 ∪ TP  // add TP into Pi+1
5.   return Pi+1
End
```

**Example 5**

Given the same S, *pa_i*, *pa_r*, and *pa_f*, as in Example 2, to find all approximate repeating patterns by applying our approach `find_approxi_pattern`.

First, we determine three cuts as in Example 3. The following processes are preceded by a level-wise manner, as shown in Figure 3. Through scanning the data sequence S once, we have the candidate set $C_1$. By checking their repeating frequencies, the patterns of length one are derived. As for the next level, we first derive the candidate set $C_2$, followed by the minimal repeating frequency checking. For each pair of patterns from $P_1$, we apply the *PJ* operators to derive $C_2$. For each pattern in $C_2$, we check its repeating frequency to determine the patterns of length two, $P_2$. Similar processes are repeated until all the patterns whose length is specified in *pa_i* are obtained.

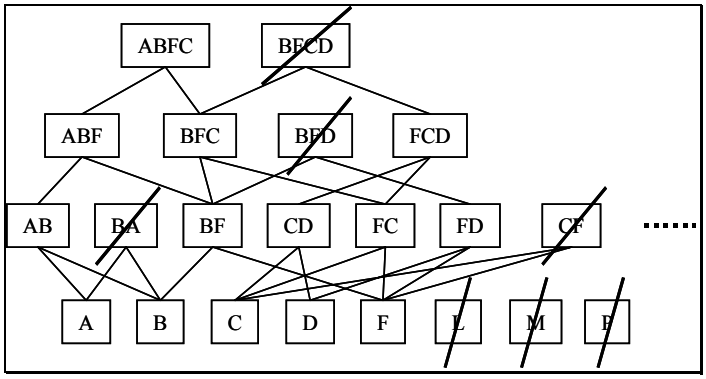


**Figure 3: The illustration of processing steps in Example 5.**

## 4. CONCLUSION

In this paper, the application of feature extraction is first presented to motivate our research on finding approximate repeating patterns from sequence data. In Section 2, followed by the definitions of match operator and approximation type, we consider the type of longer_length approximation as the fundamental problem. We develop a level-wise approach to the problem of finding approximate repeating patterns with respect to the longer_length approximation. In addition, we extend the basic approach for efficiently finding long patterns. We also complete the preliminary investigation of performance study, in which we explore the four factors having impact on the performance and show that our approach is efficient. Likewise, the refined methods and performance study are not covered in this paper because of space limitation.

The future work includes the following. First, the extensive experiments of effectiveness study on real data are still carrying on. Given a corpus of music data, our approach will be applied to discover the prototypical melody of music data, as shown in Figure 1. In our experiment design, the discovered features will be compared with music catalogs, such as [Barl75], to show the effectiveness. Moreover, we define the problems of other two types of approximations, *i.e.*, shorter_length and equal_length approximations. We are currently working on developing more efficient algorithms to solve the problems. Besides, our approach can be directly applied to monophonic music objects, but not polyphonic music objects. We are also working on exploring features for polyphonic music objects and developing corresponding methods.

## REFERENCE:


[Ahon99] Ahonen-Myka, H., "Finding All Maximal Frequent Sequences in Text," in *Proc. of Intl. Conf. on Machine Learning*, 1999.

[Barl75] Barlow, H. and S. Morgenstern, *A Dictionary of Musical Themes*, Crown Publishers, Inc., New York, 1975

[Dann02] Dannerberg, R. B. and N. Hu, "Pattern Discovery Techniques for Music Audio," in *Proc. of ISMIR*, 2002.

[Hsu98] Hsu, J. L., C. C. Liu, and A. L. P. Chen, "Efficient Repeating Pattern Finding in Music Databases," in *Proc. of Intl. Conf. on Information and Knowledge Management* (*CIKM'98*), 1998.

[Hsu01] Hsu, J. L., C. C. Liu, and A. L. P. Chen, "Discovering Non-trivial Repeating Patterns in Music Data," *IEEE Transactions on Multimedia*, Vol. 3, No. 3, 2001.

[Jone74] Jones, G. T., *Music Theory*, Harper & Row, Publishers, New York, 1974.

[Krum90] Krumhansl, C. L., *Cognitive Foundations of Musical Pitch*, Oxford University Press, New York, 1990.

[Meek01] Meek, C. and W. P. Birmingham, "Thematic Extractor," in *Proc. of ISMIR*, 2001.

[Narm90] Narmour, E., *The Analysis and Cognition of Basic Melodic Structures*, The University of Chicago Press, Chicago, 1990.

[Pien02] Pienimaki, A., "Indexing Music Database Using Automatic Extraction of Frequent Phrases, in *Proc. of ISMIR* 2002.

[Roll98] Rolland, P. Y., "FlExPat: A Novel Algorithm for Musical Pattern Discovery," in *Proceedings of the 12th Colloquium on Musical Informatics* (XII CIM), 1998.

[Roll99] Rolland, P. Y. and J. G. Ganascia, "Musical Pattern Extraction and Similarity Assessment," in Miranda, E. (eds.), *Readings in Music and Artificial Intelligence*, New York and London: Gordon & Breach - Harwood Academic Publishers, 1999.

[Self98] Selfridge-Field, E., "Conceptual and Representational Issues in Melodic Comparison," in Hewlett, W. B. and E. Selfridge-Field (eds.), *Melodic Similarity: Concepts, Procedures, and Applications* (*Computing in Musicology: 11*), The MIT Press, 1998.

[Shih01] Shih, H.-H., S. S. Narayanan, and C.-C. Jay Kuo, "Automatic Main Melody Extraction from MIDI Files with a Modified Lempel-Ziv Algorithm," in *Proc. of Intl. Symposium on Intelligent Multimedia, Video and Speech Processing*, 2001.