An Approximate String Matching Algorithm for Content-Based Music Data Retrieval.

Conference Paper · July 1999

DOI: 10.1109/MMCS.1999.779244 · Source: DBLP

3 authors:

Chih-Chin Liu
Chung Hua University
34 PUBLICATIONS **756** CITATIONS
SEE PROFILE

Jia-Lien Hsu
Fu Jen Catholic University
32 PUBLICATIONS **604** CITATIONS
SEE PROFILE

Arbee L. P. Chen
National Tsing Hua University
179 PUBLICATIONS **3,441** CITATIONS
SEE PROFILE

An Approximate String Matching Algorithm for Content-Based Music Data Retrieval

Chih-Chin Liu, Jia-Lien Hsu and Arbee L. P. Chen

Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C., E-mail: alpchen@cs.nthu.edu.tw

Abstract

In this paper, an approach for content-based music data retrieval is proposed. In this approach, thematic feature strings, such as melody strings, rhythm strings, and chord strings are extracted from the original music objects and treated as the meta data to represent their contents. The problem of content-based music data retrieval is then transformed into the string matching problem. A new approximate string matching algorithm is also proposed which provides fault tolerance ability according to the music characteristics. To show the efficiency of the algorithm, a set of experiments are performed to compare with the agrep and the fgrep utility on both synthetic and real music data.

1. Introduction

As the progress of network and data compression technologies, users can easily access a large amount of multimedia data through the internet. It is strongly required to provide users the ability to retrieve multimedia data by content. Recently, many researchers focus on the problem of content-based retrieval of image data [4][22][26] and video data [9][11][21][25][28]. However, less attention was received to the audio data. The conventional keyword-based retrieval of audio data attaches a set of keywords to the audio objects; users then pose queries against these keywords to retrieve the audio objects. Each audio object is treated as an opaque large object and the information within the audio object is omitted. The research issues have to be addressed to support content-based retrieval of audio data such as developing an indexing mechanism to extract audio features from the audio data, organizing the audio features with a suitable data structure and designing an efficient query processing algorithm to increase the performance, and providing friendly audio query interface.

The related works in the audio databases are surveyed in the following. Wold *et al*. [29] classified audio objects according to their acoustical features such as loudness, pitch, and bandwidth and map them into feature vectors in the feature space. Given an audio query example, users can find the audio objects with similar acoustical features. However, acoustical features are low level audio characteristics whose semantics is difficult to understand for non-experienced users. Lohr and Rakow [17] developed functions needed to store and manipulate audio data. Many useful operations for the audio data such as dynamic compression and low-pass filtering were provided. However, the functions needed to index and retrieve audio data are not discussed.

Previous contributions in the field of music databases are introduced as follows. For the explanation of the music terminologies used in this paper, please refer to [27]. Ghias *et al*. [8] proposed an approach for modeling the content of music objects. In this approach, the *melody* of a music object is represented as a sequence of three letters S (same *pitch*), U (pitch increased), and D (pitch decreased). The problem of music query processing is then transformed into the substring matching problem. However, this representation method is very rough and other important music features are not included. Moreover, the similarity measure of their substring matching algorithm does not take music characteristics into consideration. Prather [24] proposed a linked list data structure to store the *scores* of music data. By applying *harmonic analysis method* to traverse the scores, the *chord* information can be derived. However, how to provide users with a friendly music query interface and an efficient music query processing technique is not discussed. In [5], we used chord to represent the content of a music object. Each music object in the database and the music query are transformed into a sequence of chords. Since users who are not expert in music may not be able to precisely specify music queries, certain errors should be allowed. That is, a fault tolerance ability has to be provided. In our approach, the music query consisting of the harmonic notes with certain errors will be transformed into a correct chord, the fault tolerance ability was equipped.

In this paper, we generalize the notion of chord to thematic features and propose an efficient approximate string matching algorithm which takes the music similarity measure into consideration. The rest of the paper is organized as follows. Section 2 provides an overview of the proposed approach. The performance analysis of the approximate string matching algorithm is provided in Section 3. Finally, Section 4 concludes this paper.

2. Overview of the Approach

Due to the space limitation, the formal algorithms are skipped. In this paper, we use a music query example to motivate our approach.

Assume there are two music objects M1 and M2 in the music database whose melody strings are "sol-mi-mi-fa-re-re-do-re-mi-fa-sol-sol-sol" and "do-mi-sol-sol-re-mi-fa-fa-do-re-re-mi". The melody string of the music query Q is "do-re-mi". To find whether M1 and M2 contain the melody string, substring matching algorithm should be applied. For exact string matching, there are two well-known algorithms, *i.e.*, the KMP algorithm [13] and the Boyer-Moore algorithm [3]. Both these two algorithms are optimized. However, they need to retrieve all entire melody strings in the database for processing Q. Therefore, the first requirement for developing a string matching algorithm is to reduce the amount of data needed to retrieve from the database.

Since we cannot expect users to precisely specify music query examples, fault tolerance ability is required for content-based music data retrieval. Traditional approximate string matching algorithms [2][30][6] are designed for document searching which take *editing distance* as the similarity measure. However, editing distance cannot correctly represent the similarity considering the music characteristics. For example, the substrings "do-mi", "re-mi", and "do-re" of the melody string "do-mi-sol-sol-re-mi-fa-fa-do-re-re-mi" all are similar to the melody string "do-re-mi" with editing distance one. However, according to the *root note rule* and the *harmonic property* [12][14][18][23], "do" is more important and "re" is less important. Therefore the similarity degree should be "do-mi" > "do-re" > "re-mi". Thus, the second requirement is that the approximate string matching algorithm should also provide *music similarity measures* (to be discussed in subsections 2.2 and 2.3).

To satisfy the first requirement, we try to retrieve only the notes involved in the query string from the melody strings. To achieve this, we first store positions of the same notes in a melody string into linked lists as shown in Figure 1(a). Each element in the linked lists is of the form (x;y) which denotes the y-th note of the melody string of the x-th music object in the database. The number of the linked lists is the same as the size of the alphabet of the melody strings which is assumed seven, *i.e.*, "do", "re", "mi", "fa", "sol", "la", and "si". The three linked lists corresponding to "do", "re", and "mi" involved in Q are retrieved with two dummy nodes *start* and *end* as shown in Figure 1(b). The order of the three linked lists is the same as the sequence of their corresponding symbols appearing in Q. In this case, only 15 nodes out of the total 25 nodes need to be retrieved. In general, only $O(nm/A)$ nodes need to be retrieved, where n is the number of nodes in the melody string of the query, m is the total number of nodes of all melody strings in the database, and A is the alphabet size of the melody strings.

To search for the substrings of the melody strings which match the query string "do-re-mi", we (1) find the substrings of the melody strings "do-re" and "re-mi" and (2) merge these substrings to find the occurrences of the substrings "do-re-mi". When we merge the substrings "do-re" and "re-mi" into "do-re-mi", we have to make sure that the "re's" in both substrings are the same one in the same melody string. (1) is done by attaching an *exact link* between two elements in the retrieved linked list, whose corresponding notes in the feature string form such a substring "do-re" and "re-mi". (2) is done by traversing the directed graph formed by these exact links to generate the paths whose corresponding substrings match "do-re-mi". The rules for constructing the exact links and their extensions for the approximate string matching will be illustrated in the following.

2.1. Exact String Matching

The exact string matching algorithm goes as follows. Let pos(ni) represent the position of ni in the related melody string. For each pair of nodes (n1, n2) taken from two adjacent linked lists of the same melody string, if pos(n1) + 1 = pos(n2), which means "n1-n2" is a substring of the melody string, we build an *exact link* form n1 to n2 as shown in Figure 1(c). We also build an exact link from the *start* node to n_i if n_i has an outgoing exact link and from n_i to the *end* node if n_i has an incoming exact link as shown in Figure 1(d). The music similarity measure corresponding to an exact link is set to one since the corresponding substring is the same as the corresponding substring of the query. By traversing the exact links from the *start* node to the *end* node, we find the query matches the substring started from the 7th position to the 9th position of the melody string of the music object M1. However, M2 does not contain any substring which matches the query.

2.2. Approximate String Matching

To provide fault tolerance ability, three kinds of errors, *i.e.*, dropout errors, insertion errors, and transposition errors should be allowed. For example, for the query string "do-re-mi", the melody strings "do-re", "re-mi", and "do-mi" approximately match it with one dropout error; the melody string "do-fa-re-mi" and "do-re-sol-mi" approximately match it with one insertion error; and the melody strings "do-re-?", "?-re-mi", and "do-?-mi" approximately match it with one transposition error. However, although the editing distance of these melody strings are the same, their music similarity measures may be different. We use the function sim(n), to denote the similarity measure when "n" is dropped out, inserted, or transposed from the melody string. The computation of sim(n) will be discussed in the next subsection.

The first procedure of the approximate string matching algorithm is the same as the exact one. The next step is to find the possible dropout errors. We note that the value of the second element in the "do" linked list is (2;1) and the value of the fourth element in the "mi" linked list is (2;2), which means there is a substring "do-mi" in the music object M2. A dropout error occurs and we build a dropout

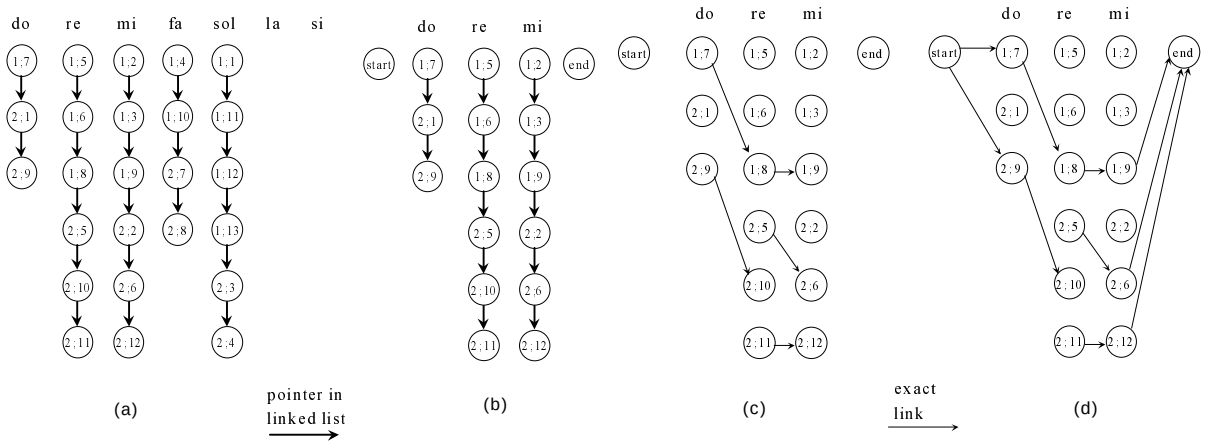

Figure 1: Exact string matching example.

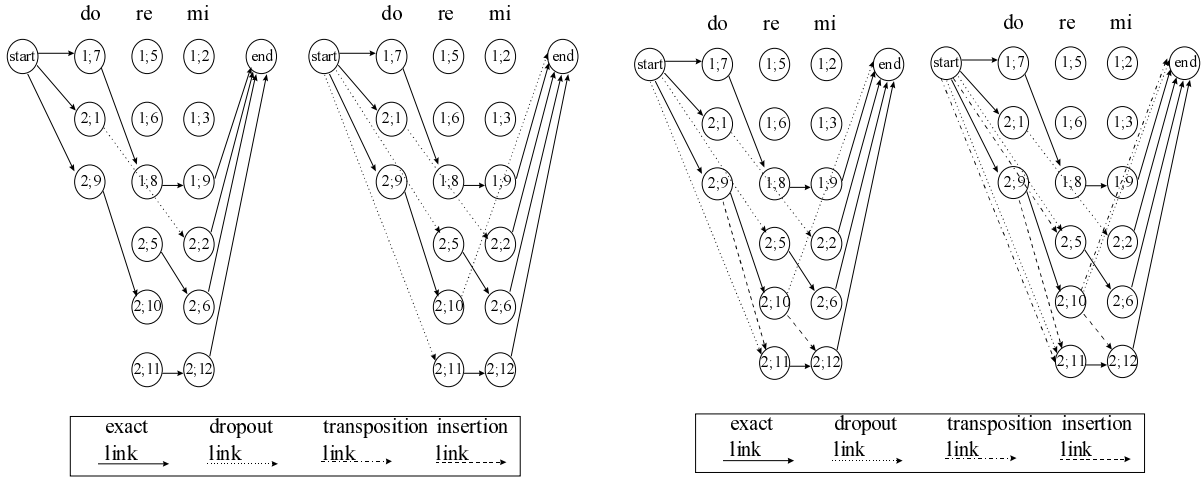

Figure 2: Approximate string matching example.

link between these two nodes to denote it. The result is shown in Figure 2(a). If a dropout error occurs at the first node, a substring "re-mi" is contained in the melody string. We build a dropout link between the start node and the related node in the "re" linked list. If a dropout error occurs at the latest node, a substring "do-re" is contained in the melody string. We build a dropout link between the related node in the "re" linked list and the end node as shown in Figure 2(b).

The third step of the approximate string matching algorithm is to find the possible insertion errors. We note that the value of the fifth element in the "re" linked list is (2;10) and the value of the sixth element in the "mi" linked list is (2;12), which means there is a substring "re-?-mi" in the melody string of the music object M2. An insertion error occurs and we build a *insertion link* between these two nodes to denote it. Similarly, we build an insertion link from the third element in the "do" linked list to the sixth element in "re" linked list. The result is shown in Figure 2(c).

The fourth step of the approximate string matching algorithm is to find the possible transposition errors. We note that the value of the fourth element in the "re" linked list is (2;5), which is not the first symbol of the melody string of the music object M2, and the value of the fifth element in the "mi" linked list is (2;6), which means there is a substring "?-re-mi" in the melody string of the music object M2. A transposition error occurs and we build a *transposition link* between these two nodes to denote it. Similarly, we can build the transposition links for all transposition errors of the forms "?-re-mi", "do-?-mi", and "do-re-?". The result is shown in Figure 2(d).

To derive the query results of this music query, we traverse the directed graph formed by these four kinds of links to generate all the paths from the *start* node to the *end* node. The query results are shown in Figure 3. The method for generating the music similarity measures corresponding to these paths will be detailed explained in next subsection.

2.3. Music Similarity Measures

A music similarity measure is defined to be a number between 0 and 1. For each link in the directed graph, the larger the music similarity measure is, the more similar the corresponding feature substring is to the music query substring. Note that the music similarity measure of an exact link is one which means the substring corresponding to the exact link exactly matches the music query substring.

Consider the melody string M = "do-re-mi-fa-sol-la-si", there are seven melody strings similar to M with one dropout error, *i.e.*, "re-mi-fa-sol-la-si", "do-mi-fa-sol-la-si", "do-re-fa-sol-la-si", "do-re-mi-sol-la-si", "do-re-mi-fa-la-si", "do-re-mi-fa-sol-si", and "do-re-mi-fa-sol-la". The music similarity measures corresponding to the seven dropout errors are denoted as sim("do"), sim("re"), sim("mi"), sim("fa"), sim("sol"), sim("la"), and sim("si"). However, from the note "do" point of view, according to the music theory [12][14][18][23][27], the following relationships exist: "do" is the *root note*, "sol" is the *dominant* and the *fifth note*, "fa" is the *subdominant*, "si" is the *leading note*, and "mi" is the *third note*. Therefore, we can assume sim("do") = 0.1, sim("re") = 0.8, sim("mi") = 0.5, sim("fa") = 0.6, sim("sol") = 0.3, sim("la") = 0.8, and sim("si") = 0.7. The chord strings also possess similar music characteristics in which the chords with the *closely related keys* are more similar than other chords.

For a path P of length n, if the path consists of e exact links, d dropout links, i insertion links, and t transposition links, $e + 2d + i + 2t = n + 1$ exists. This is because a dropout link and a transposition link skip a symbol in the music query string. Let $sim(D_k)$, $sim(I_k)$, $sim(T_k)$, and sim(P) denote the music similarity measures of the dropout link D_k, the insertion link I_k, the transposition link T_k, and the path P, respectively. sim(P) can be computed by the following formula:

$$sim(P) = \frac{e + d + t + \sum_{k=1}^{d} sim(D_k) + \sum_{k=1}^{i} sim(I_k) + \sum_{k=1}^{t} sim(T_k) \quad - 1}{n}$$

The formula is explained as follows. The similarity measure of the path P is the summation of the similarity measures of the four kinds of links in P, which equals $e + \sum_{k=1}^{d} sim(D_k) + \sum_{k=1}^{i} sim(I_k) + \sum_{k=1}^{t} sim(T_k)$. However, since a dropout link and a transposition link skip a symbol in the music query string, for a path including d dropout links and t transposition links, there are d + t symbols skipped. We need to add $d + t$ to the similarity measure. Further, since $e + 2d + i + 2t = n + 1$, we should subtract one from the similarity measure. Finally, the similarity measure is divided by n to normalize its value into the range between 0 to 1. For example, the music similarity measure of the first path in Figure 4 is sim(P1) = (4 – 1) / 3 = 1; the music similarity measure of the second path is sim(P2) = (2 + 1 + 0.8 – 1) / 3 = 0.933, where sim("re") = 0.8; and the music similarity measure of the third path is sim(P3) = (2 + 1 + 0.1 – 1) / 3 = 0.7, where sim("do") = 0.1.

3. Performance Analysis

To show the efficiency and feasibility of our exact/approximate string matching algorithms, we perform a series of experiments and compare our approach with the *agrep* and the *fgrep*. The reason for choosing *fgrep* as the comparing target is that *fgrep* is the fast and standard string matching algorithm which is widely available on the UNIX systems. And, *agrep* is a new tool for text searching allowing errors [30]. Since *agrep* and *fgrep* cannot perform approximate string matching with music similarity measures, for approximate string matching cases, we only compare with our exact string matching algorithm. There are two running data sets. One set of music objects is generated randomly. The other set of music objects contain real music objects of MIDI form [20]. All music objects are parsed to extract their melody strings and rhythm strings. All experiments are performed on the melody strings.

3.1. Exact String Matching

Figure 4 and Figure 5 illustrate the execution time versus the length of the query string for synthetic music data set and real music data set, respectively. In both cases, the execution time of *fgrep* is about the same, and the execution time of *agrep* grows rapidly as query string length is more than 6, shown in Figure 4 and Figure 5. The time complexity of *fgrep* is $O(m+n)$, where m is the length of the query string and n is the length of the feature string [19]. In this experiment, since the length of feature strings (greater than 88000) is much larger than the length of query strings (less than 20), the performance of *fgrep* depends on the length of feature strings. On the other hand, the time complexity of our string matching algorithm is $O(mn/A)$, the execution time of our approach linearly increases with the length of the query string. Moreover, since our algorithm only retrieves the linked lists involved in the query string from the music index, *i.e.*, n/A is much less than one, better performance is achieved than that of *fgrep*. As we have assumed, n will not be greater than 20. However, when n is larger than A, all linked lists have to be retrieved for processing and the performance of our algorithm will be worse than that of the utility *fgrep*.

3.2. Approximate String Matching

In this group of experiments, we discuss the approximate string matching cases. There are three cases: the dropout, transposition and insertion cases. For the experiment for showing the effects of the length of query strings, we assume that the number of the music objects is 400, the average length of the feature strings of the music objects is 2290 for real music data set. The performances of

both experiments are measured by the average execution time of 500 music queries. Figure 6 show the experiment results. The execution times for the approximate string matching algorithms are more than that of the exact string matching since the construction of the dropout, transposition and insertion links needs extra time. However, the time complexities of the exact string matching algorithm and the approximate string matching algorithms are the same.

4. Conclusion

In this paper, we propose an approximate string matching algorithm for content-based music data retrieval. In our approach, we first extract music feature strings, such as melody strings, rhythm strings, and chord strings, from the original music objects and store them in the linked list form. These feature strings are used as the music index for content-based music data retrieval. A new approximate string matching algorithm is then proposed to match the feature strings of music queries with the feature strings of the music objects in the music database. In our algorithm, only the linked lists involved in the music query are needed to be retrieved for processing. Furthermore, our algorithm provides fault tolerance ability and uses the music similarity measure to compute the similarity of two feature strings.

Future research includes the following issues. First, the dropout links, the insertion links and the transposition links proposed in this paper can only represent errors with editing distance one. We shall consider errors with higher editing distances. Second, to show the practical effectiveness of our algorithms, an extensive performance evaluation of the Muse system is being carried out.

References:

[1] Aho, A. V. and M. J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *CACM*, Vol. 18, No. 6, pp.333-340, 1975.

[2] Baeza-Yates, R. and G. H. Gonnet, "A New Approach to Text Searching," *CACM*, Vol. 35, No. 10, Oct. 1992, pp. 74-82.

[3] Boyer, R. S. and J. S. Moore, "A Fast String Searching Algorithm," *CACM*, Vol. 20, Oct. 1977.

[4] Chiueh, T. C., "Content-Based Image Indexing," in *Proc. of the 20th VLDB Conf.*, pp. 582-593, 1994.

[5] Chou, T. C., A. L. P. Chen, and C. C. Liu, "Music Databases: Indexing Techniques and Implementation," in *Proc. of IEEE Intl. Workshop on Multimedia Data Base Management System*, 1996.

[6] Corman, T.H., C.E. Leiseson and R.L. Rivest, *Introduction to Algorithms*, The MIT Press: McGraw-Hill, 1993.

[7] Fan, J. J. and K. Y. Su, "An Efficient Algorithm for Matching Multiple Patterns," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 5, No.2, pp. 339-351, 1993.

[8] Ghias, A., J. Logan, D. Chamberlin, and B. C. Smith, "Query by Humming: Musical Information Retrieval in an Audio Database," in *Proc. of ACM Multimedia*, 1995, pp. 231-236.

[9] Hjelsvold, R. and R. Midstraum, "Modelling and Querying Video Data," in *Proc. of VLDB Conf.*, pp. 686-694, 1994.

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

[11] Jain, R., "Metadata in Video Databases," *SIGMOD RECORD*, Vol. 23, No. 4, pp. 27-33, Dec. 1994.

[12] Jones, M. R. and S. Holleran, *Cognitive bases of musical communication*, American Psychological Association, 1991.

[13] Knuth D. E., J. H. Morris, V. R. Pratt, "Fast Pattern Matching in Strings," *SIAM J. Comput.*, pp.323-350, 1977.

[14] Lerdahl, F. and R. Jackendoff, *A generative theory of tonal music*, The MIT Press, 1983.

[15] Liu, C. C., J. L Hsu, and A. L. P. Chen, "Efficient Near Neighbor Searching Using Multiple Indexes for Content-Based Multimedia Retrieval," *Multimedia Tools and Applications* (to be appeared).

[16] Liu C. C., J. L. Hsu and A. L. P. Chen, "Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Databases," in *Proc. of IEEE International Conf. on Data Engineering*, 1999.

[17] Lohr, M. and T. C. Rakow, "Audio Support for an Object-Oriented Database-Management System," *ACM Multimedia Systems Journal*, pp. 286-297, 1995.

[18] Lundin, R. W., *An objective psychology of music*, Robert E. Krieger Publishing Company, 1985.

[19] Marshall, Kirk, McKusick et al., *The design and implementation of the 4.4BSD operating system*, Addison-Wesley, 1996.

[20] MIDI Manufacturers Association (MMA), *MIDI 1.0 Specification*, http://www.midi.org/.

[21] Oomoto, E. and K. Tanaka, "OVID: Design and Implementation of a Video-Object Database System," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, No. 4, pp. 629-643, 1993.

[22] Petrakis, E. G. M. and S. C. Orphanoudakis, "Methodology for the Representation, Indexing and Retrieval of Images by Content," *Image and Vision Computing*, Vol. 11, No. 8, pp. 504-521, 1993.

[23] Pierce, J. Robinson, *The science of musical sound*, W. H. Freeman and Company, 1992.

[24] Prather, E., "Harmonic Analysis from the Computer Representation of a Music Score," *Communication of the ACM*, Vol. 39, No. 12, Dec. 1996, pp. 119 (pp. 239-255 of *Virtual Extension Edition of CACM*).

[25] Smoliar and H. Zhang, "Content-Based Video Indexing and Retrieval," *IEEE Multimedia*, Vol. 1, No. 2, pp. 62-72, 1994.

[26] Srihari, "Automatic Indexing and Content-Based Retrieval of Captioned Images," *IEEE Computer*, pp. 49-56, Sep. 1995.

[27] Thomsett, M. C., *Musical Terms, Symbols, and Theory: An Illustrated Dictionary*, St James Press, 1985.

[28] Weiss, R., A. Duda, and D. K. Gifford, "Composition and Search with a Video Algebra," *IEEE Multimedia*, pp. 12-25, Spring 1995.

[29] Wold, E., et al., "Contend-Based Classification, Search, and Retrieval of Audio," *IEEE Multimedia*, Vol. 3, No. 3, Fall 1996, pp. 27-36.

[30] Wu, S. and U. Manber, "Fast Text Searching Allowing Errors," *CACM*, Vol. 35, No. 10, Oct. 1992, pp. 83-91.

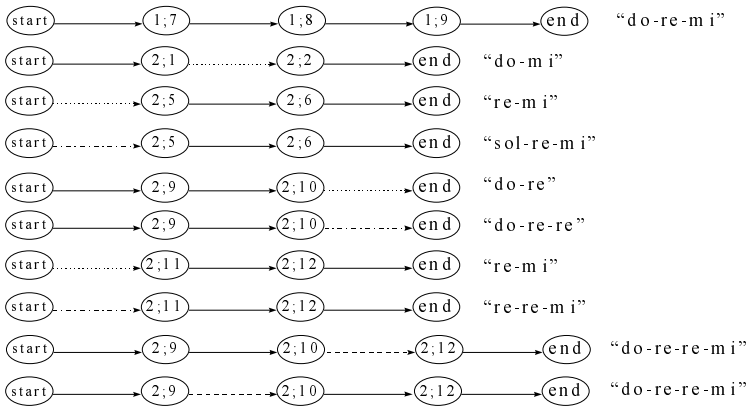

Figure 3: Approximate string matching results.

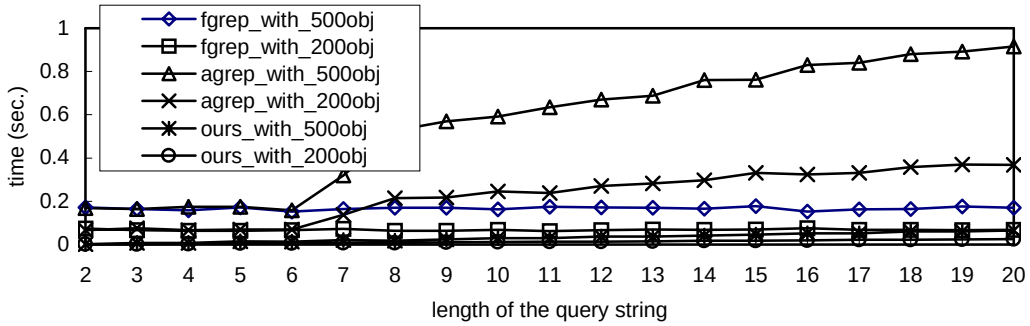

Figure 4: Execution time vs. length of the query string (synthetic music data set).

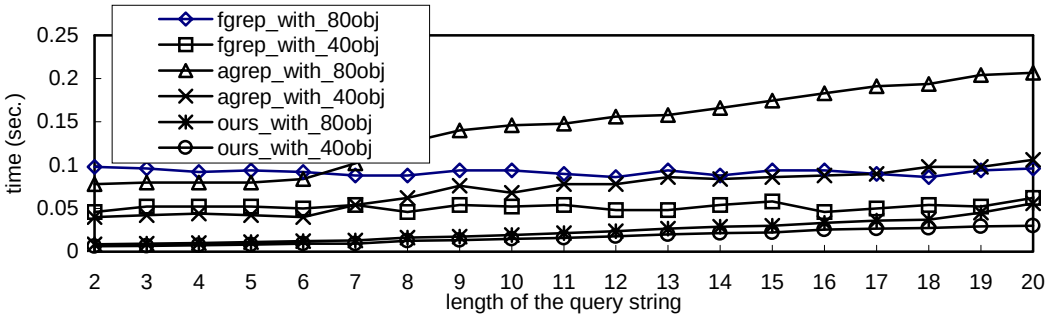

Figure 5: Execution time vs. length of the query string (real music data set).

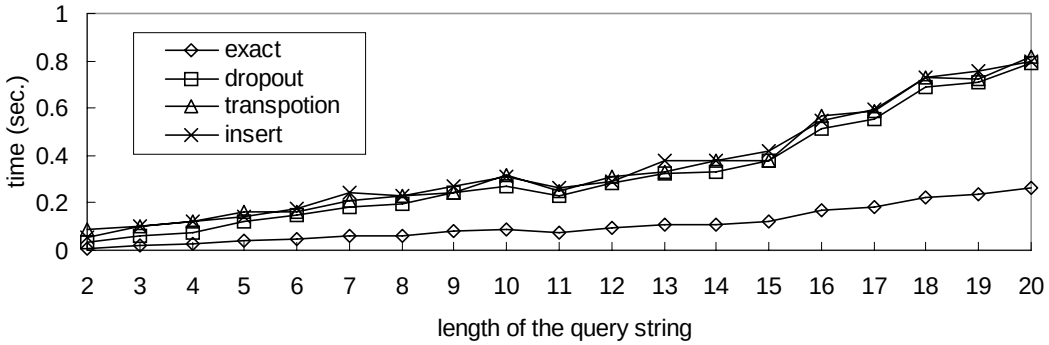

Figure 6: Execution time vs. length of the query string (real music data set).