Multimedia Systems (2008) 14:51–70
DOI 10.1007/s00530-007-0111-z

REGULAR PAPER

Continuous query processing over music streams based on approximate matching mechanisms

Hung-Chen Chen · Yi-Hung Wu · Yu-Chi Soo · Arbee L. P. Chen

Published online: 6 December 2007

Abstract It is foreseen that more and more music objects in symbolic format and multimedia objects, such as audio, video, or lyrics, integrated with *symbolic music representation* (SMR) will be published and broadcasted via the Internet. The SMRs of the flowing songs or multimedia objects will form a *music stream*. Many interesting applications based on music streams, such as interactive music tutorials, distance music education, and similar theme searching, make the research of content-based retrieval over music streams much important. We consider multiple queries with error tolerances over music streams and address the issue of approximate matching in this environment. We propose a novel approach to continuously process multiple queries over the music streams for finding all the music segments that are similar to the queries. Our approach is based on the concept of n-grams, and two mechanisms are designed to reduce the heavy computation of approximate matching. One mechanism uses the clustering of query n-grams to prune the query n-grams that are irrelevant to the incoming data n-gram. The other mechanism records the data n-gram that matches a query n-gram as a partial answer and incrementally merges the partial answers of the same query. We implement a prototype system for experiments in which songs in the MIDI format are continuously broadcasted, and the user can specify musical segments as queries to monitor the music streams. Experiment results show the effectiveness and efficiency of the proposed approach.

Keywords Music stream · Continuous query processing · Event stream · Approximate matching · Lower bounding

H.-C. Chen · Y.-C. Soo
Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan, ROC
e-mail: Jesse@cs.nthu.edu.tw

Y.-C. Soo
e-mail: g934370@oz.nthu.edu.tw

Y.-H. Wu
Department of Information and Computer Engineering, Chung Yuan Christian University, Chung Li 32023, Taiwan, ROC
e-mail: yhwu@cycu.edu.tw

A. L. P. Chen (✉)
Department of Computer Science, National Chengchi University, Taipei 11605, Taiwan, ROC
e-mail: alpchen@cs.nccu.edu.tw

1 Introduction

Recent researches [4,5] revealed the importance of integrating the *symbolic music representation* (SMR) with multimedia objects for the new music-related applications and projects, such as music education and interactive entertainment. Moreover, the composers are used to producing their works by writing down the scores, and the symbolic format can represent the scores in a more precise and musical way than other formats. To distribute, share, and receive music in a symbolic format is convenient for the content distributors, musicians, and users to interchange, annotate, and edit the music objects. Therefore, it is foreseen that more and more music objects in symbolic format and multimedia objects, such as audio, video, or lyrics, integrated with SMR will be published and broadcasted via the Internet for various services. Based on SMR, various approaches [46] have been proposed to provide an efficient and effective service for content-based music retrieval. The key issues of these approaches include the music representations, index structures, query processing methods, and similarity measurements. While most researchers focus on content-based retrieval for static music databases, we concentrate our attention on content-based retrieval in the streaming environment.

Various applications can adopt the approach of content-based retrieval in the streaming environment. Consider the applications of interactive music tutorials and distance education at music school. The students may be requested to compose their own music works by utilizing the popular themes and phrases of the classical masterpieces in the Baroque ages. The students can play their works on the piano or other instruments, or write down their scores. The notes played by a student or the written scores will be transformed into the symbolic format, such as MIDI files [37] or MusicXMLs [41], and then continuously sent to the server for examining whether the work of the student satisfies the teacher's requirements. The server works with a query database that contains the themes and phrases that the teacher asks for as well as a knowledge database that contains the music theories derived from the original classical masterpieces. For each incoming music work, the server can identify which themes and phrases in the query database are included. Furthermore, the knowledge of the original classical masterpieces, such as music structure, chord, and composing techniques is then compared with the music work of the student. The results of the comparison will be immediately returned to the students so that they can further improve their works based on the related music theories. The teacher will also receive the music work of a student as well as the corresponding results to further guide the student by annotating and editing this music work. There could be a lot of music works of the students continuously flowing into the server. How to efficiently deal with the flowing music works will be an important issue.

Consider the applications of similar theme searching and music copy detection for composers, users, and publishers. The most important thing the composers care about is that whether their works are similar to other existing works. The composers can issue the original themes and phrases on the website for investigation. The website will confirm whether the incoming music work contains any published themes or phrases. Since there could be thousands of new music work produced each day, the website may require an efficient approach to process the incoming music works. Similarly, the users who listen to the music channels via Internet can send specific music segments as queries for searching interesting songs. Publishers can also monitor the themes of their works by continuously checking the scores of the multimedia objects whether passing through the selected routers or being shared via websites. The streaming environment could be formed with the songs broadcasted via the Internet or the multimedia objects passing through the selected routers, servers, and websites. The SMRs of the flowing songs or multimedia objects will form a *music stream*. Therefore, the problem to be solved in this paper is regarded as content-based retrieval over music streams. To request content-based retrieval over music streams, the user can issue the interesting music segments as the queries. Such a music segment can be the incipit or refrain of the music object. As time goes by, the user will receive the notifications from the system when a music segment close to the previously issued query occurs on the music stream. Furthermore, the multimedia object containing the interesting music segment can be continuously identified as the music stream flows.

Various problems in the streaming environment have been discussed in the literature [20]. Data in new applications, such as web accesses, financial tickers, network packets, and sensor data [13,56], are all in the form of continuous streams, called *data stream*. The music stream that we deal with is one kind of data stream. To support the new applications, *continuous query* (CQ) is designed to allow the users to get new results from a data stream without having to issue the same query repeatedly. Most of the data stream management systems (DSMSs) [7,11,24,40] support the CQs that have SQL-like syntax and the enhanced support for windows [10]. However, the SQL-based CQ cannot express the queries well in the form of music segments. Therefore, we design another kind of CQ for the problem of content-based retrieval over music streams.

To keep the property of continuity for music segments, we design a new kind of CQ, named the *sequence query* (SQ), as a sequence of events, where an event denotes the set of notes played at the same time [16,32]. This pitch representation is also named *homophonic reduction* [46], assuming the independence among the notes with overlapping duration. The pitch information is the most perceptual property in the SMR for the users [15]. In pitch representation, the pitch values are non-negative integers and smaller than 128 following the MIDI standard [37], which is the most popular SMR. For example, in the SQ specified as $\langle 60\rangle\langle 64, 67\rangle\langle 62\rangle\langle 65, 69\rangle$, the second event $\langle 64, 67\rangle$ indicates that the notes with pitches 64 and 67 in the MIDI format are played simultaneously. Similarly, the music stream can be regarded as an *event stream*, which is a continuous and infinite sequence of events. For an SQ, the answers will be all the data segments on the event streams that are exactly the same as the SQ itself. That is, the exact answers of a SQ must preserve the order between the events of the SQ as well as the content in each event. However, in most of cases, exact answers cannot satisfy the users' requirements. There are several reasons to provide the answers that are similar to the SQ for the user, not only the exact answers. First, it is often difficult for the user to precisely specify what she/he wants to find on the music streams. Second, a music work may become a variation by adding some grace notes. Third, a familiar theme could be adopted by different music pieces with some variations. Therefore, it is necessary for the user to specify an SQ followed by an error bound for approximate matching. The error bound indicates the maximal allowable difference between the SQ and the qualified answer on the event stream.

For approximate matching, we have to determine a suitable distance measure first. In this paper, we adopt the *edit distance* as the distance measure. The edit distance [27,42] is commonly used in string matching to measure the difference between two strings. Various applications, such as music retrieval, root-cause analysis, speech recognition, plagiarism detection, and DNA alignment [32,38,43,50], use the edit distance as their similarity measures. For the problem we consider, the edit distance is intuitive for revealing the difference between two musical segments and is simple for users to understand. Incorporating specific music characteristics, such as tonality or strong beats in the distance measure may make the measurement of the similarity more accurate; however, it has not yet been proved. Therefore, in this paper we focus on the basic form of edit distance and treat further evaluation of the returned results as an option of post-processing. The edit distance between two strings is the minimum cost of editing operations, i.e., insertion, deletion, and replacement, required to transform one string into the other [14]. Based on the edit distance, given a query and an error bound, a data segment on the event stream is defined as an *approximate answer* of the query if the distance does not exceed the error bound. Similarly, given a data segment on the event stream, we also define that a query is an *approximate match* of the data segment under the same condition. In summary, we use the edit distance as the similarity measure and formulate our problem as follows:

Given a set of SQs, each has its own error bound, how to continuously monitor the event streams and report all the segments that are approximate answers of certain SQs as soon as the segments arrive?

In some circumstances, one music piece may be reported redundantly if it contains more than one music segment close to the same query. For the application of searching interesting songs, we simply skip the redundant notifications for the user. For other applications such as similar theme searching and music copy detection, it will be necessary to reveal all the segments that are close to the query in order not to miss any dubiety.

In this paper, we develop a novel approach for content-based retrieval over music streams to deal with the proposed problem that has not yet been solved. To the best of our knowledge, our approach is the first one to consider this problem. Our approach is based on the assumption that each music piece or multimedia object is accompanied with its SMR. Recent music-related applications and projects [4,5] support this assumption. Our approach addresses the two new issues that the traditional approaches of content-based music retrieval never encounter, i.e., simultaneous processing of multiple queries and unbounded amount of data. For the streaming environment, we adopt the concept of n-gram indexing so that the SQs can be organized in an efficient way, especially when the lengths of the SQs are different. Due to the essential difference between static databases and streaming environment, no existing n-gram indexing methods [47,49,57] can solve our problem. Moreover, most of them are designed for exact matching instead of approximate matching. We develop several techniques to reduce the computation cost by sharing the computation among SQs, reducing the number of data segments to be checked, and keeping the intermediate results for reuse. These techniques are designed based on the characteristics of n-gram indexing and will be further discussed in Sect. 2. Formal proofs are also provided to demonstrate the correctness of the proposed techniques. Therefore, our approach guarantees that no approximate answer of an SQ will be lost. Experiment results show the effectiveness and the efficiency of our approach on a prototype system in which real songs in the MIDI format are continuously broadcasted.

The rest of the paper is organized as follows. In Sect. 2, the overview of our approach for multiple SQs processing on the event stream is provided. The three major components, i.e., query manager, pruning mechanism, and merging mechanism, and the associated properties are presented in Sects. 3, 4, and 5, respectively. To evaluate the effectiveness and efficiency, we build a prototype system to provide the service of searching music segments over music streams by our approach. The experiment results in Sect. 6 show that our approach performs well under the real-time and scalability requirements. In Sect. 7, the related works are discussed. Finally, we conclude the paper with some future works in Sect. 8.

2 Basic idea and system architecture

To introduce our approach, we first define two terms. The *length* of a query/data segment stands for the number of events in it, and the *size* of an event refer to the number of notes in it. The main idea of our approach is to design an efficient indexing method for both multiple queries processing and approximate matching. For this purpose, the concept of n-gram indexing is adopted. We decompose the SQ into disjoint parts with length n, called the *query n-grams*. On the other hand, we slide a window on the event stream to extract each data segment with length n, called the *data n-gram*. Note that the data n-grams can be partially overlapping. Comparing the data n-gram with each query n-gram is a straightforward but inefficient solution. For efficiency, our idea here is to greedily prune a set of query n-grams for an incoming data n-gram by only one step of checking. To accomplish this goal, we initially divide similar query n-grams from all SQs into a cluster and derive the *summarization* of each cluster. Then we compute the distance between the incoming data n-gram and the summarization of a cluster, which is referred to as *restricted edit distance*. The restricted

edit distance is proved shorter than or equal to the minimum distance between the data n-gram and any query n-gram in the cluster. If the restricted edit distance is larger than all the error bounds of the involved SQs in the cluster, every query n-gram in the cluster will not be an approximate match of the data n-gram and therefore can be pruned. We guarantee that no approximate answer will be lost if we discard any query n-gram for the data n-gram based on the result. Accordingly, the computation can be reduced. The formal definitions of the summarization and the restricted edit distance will be given in Sects. 3 and 4, respectively.

Since the number of data n-grams on the event stream to be checked is huge, the cost of computing the distance between a data segment and each SQ will be high. Our approach takes the advantages of the n-gram indexing to reduce the number of data segments to be checked and to keep the intermediate results for reuse as less as possible. Initially, we set the length of n-gram (the value of n) to be larger than the error bound ε of any SQ. Suppose the SQ has m query n-grams. To come out with the final answer, we have to find out each approximate answer for each query n-gram. A data segment that consists of only $(m-1)$ approximate answer of the $(m-1)$ query n-grams will exceed the error bound ε. Moreover, the m query n-grams of the SQ are in sequential order. We conclude that the approximate answer of the ith query n-gram is worthy of being recorded only if there exist $(i-1)$ data n-grams that are the approximate answers of the 1st, ... $(i-1)$-th query n-grams, respectively. This finding motivates us to develop a mechanism that can incrementally merge the approximate answers of the query n-grams of the SQ and decide whether the data segment after merging (called *partial answer*) is worthy to be maintained. Moreover, the data segments whose lengths are far from the SQ are unnecessary to be checked. The length difference between the SQ and its answer implies the minimum number of insertions or deletions in the edit distance, which cannot exceed the error bound ε, either. Therefore, for the SQ with length L, only the data segments with lengths in $[L-\varepsilon, L+\varepsilon]$ should be considered. All these techniques assist our approach to keep only the data segments that are possible to be the final answer of the SQ and guarantee that no answer will be lost.

Since the data segments on the event stream may overlap, the computation on the overlap will be redundant. To avoid redundant computation, we keep the information of the distance between a data n-gram and a query n-gram when the distance is computed for the first time. The information can be used to estimate the minimum distance between the subsequent data segment containing the data n-gram and the SQ containing the query n-gram. We guarantee that the estimated minimum distance is always equal to or less than the actual distance. Therefore, no answer will be lost if we prune the data segment when the estimated minimum distance is above the error bound of the SQ.

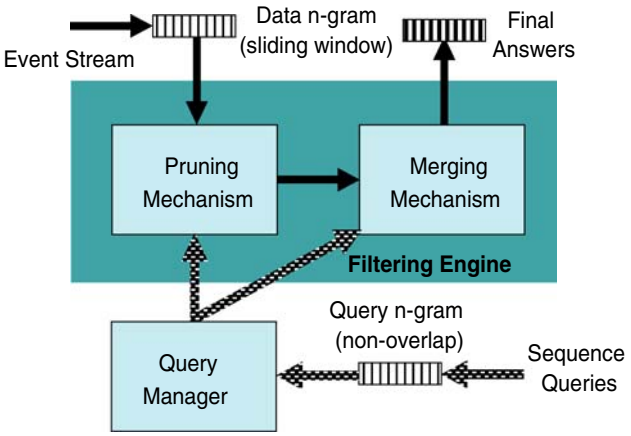

Fig. 1 The system architecture of our approach

As shown in Fig. 1, our approach consists of three components. The *query manager* consumes a set of SQs and produces the clusters constituted by their query n-grams as a result. The clusters together with their summarizations are then sent to the pruning mechanism. In addition, the query manager also creates a query buffer for each query n-gram to maintain the partial answers. As the data n-grams continuously flow in, the *pruning mechanism* takes each data n-gram to search the clusters for finding all the query n-grams that are its approximate matches. If the restricted edit distance is larger than the maximum of the error bounds in the cluster, all the query n-grams in the cluster can be skipped. Otherwise, for each query n-gram in the cluster, the data n-gram is sent to the corresponding query buffer for further processing.

For each data n-gram sent to a query buffer, the *merging mechanism* examines whether it needs to be merged with the partial answers generated from the prior query n-grams in the same SQ. The merging operation occurs only if the estimated distance between the merged result and the SQ does not exceed the error bound of the SQ. The estimated distance is obtained by using the intermediate results computed from the associated partial answer, and the currently computed distance between the data n-gram and the corresponding query n-gram. After the merging operation, a new partial answer is generated by merging the partial answer with the data n-gram, and then put into the query buffer for merging with the approximate answers of the subsequent query n-grams in the same SQ.

Note that we present the approach for one music stream in the following sections, and this approach can easily be extended for multiple music streams. Moreover, this approach can be extended to deal with the pitch variation (transposition) for music retrieval. If we want to find the pitch variations of a query, we can add the extended queries, which are the pitch variations of the original query with f semitones. The f value can be decided by the user. For example, if the query is $\langle 60\rangle\langle 64, 67\rangle\langle 62\rangle\langle 65, 69\rangle$, the

extended query with one semitone will be ⟨61⟩⟨65, 68⟩⟨63⟩⟨66, 70⟩. Note that f can also be a negative integer. This approach can also be applied to a broader area of applications by using the contour representation, i.e., the first-order interval, for polyphonic music introduced in [32]. The first-order interval collects the differences in pitch scales between two consecutive events. Accordingly, an event sequence can be transformed into a first-order interval sequence. Since a first-order interval sequence follows the definition of the event sequence, our algorithm can be directly applied.

3 Query manager

The query manager processes the set of SQs in four steps, which are separately described below:

3.1 SQ decomposition

Each SQ is decomposed into disjoint query n-grams, where the predetermined parameter n is set to be larger than the maximum among the error bounds of all the SQs. Moreover, each query n-gram is associated with a *local error bound*, which is set as the error bound of the SQ. It is possible that the length of an SQ is not the multiple of n. In that case, we append a number of special events ⟨$⟩ to lengthen the SQ into a multiple of n. The special event is regarded as an exact match of any event. For example, given $n = 3$ and SQ = ⟨a,b,c⟩⟨b,d⟩⟨a,b,d⟩⟨e,f⟩⟨a,c⟩, one special event is appended and two query n-grams ⟨a,b,c⟩⟨b,d⟩⟨a,b,d⟩ and ⟨e,f⟩⟨a,c⟩⟨$⟩ are obtained.

3.2 Query n-gram clustering

Given a system-defined threshold μ, all the query n-grams are classified into clusters so that the edit distance between two query n-grams in the same cluster does not exceed μ. We introduce the edit distance used in the following. Let a_i and b_j be the events on two sequences S_1 and S_2, respectively. Since an event is a set of values, we adopt the *Jaccard coefficient* [51] to measure the similarity between a_i and b_j, denoted as $SIM(a_i, b_j)$.

$$SIM(a_i, b_j) = \frac{|a_i \cap b_j|}{|a_i \cup b_j|} \tag{1}$$

where $|X|$ denotes the size of a set X.

Definition 1 (Edit distance) The edit distance between two even sequences S_1 and S_2, denoted as $edit\text{-}DIS(S_1, S_2)$, is the minimum cost of edit operations required to transform S_1 into S_2 [14]. Let λ denote the null character. Three edit operations and the corresponding costs [32] are defined as follows:

S_1 \ S_2		<a,b,c>	<b,d>	<a,c,d>	<e,f>	<a,e>
	0	1	2	3	4	5
<a,b,c>	1	0	1	2	3	4
<b,d>	2	1	0	1	2	3
<a,b,d>	3	2	1	1/2	3/2	5/2
<e,f>	4	3	2	3/2	1/2	3/2
<a,c>	5	4	3	7/3	3/2	7/6

Fig. 2 The matrix for computing edit-DIS(S_1, S_2)

$$\begin{aligned} &\text{Deletion: } \text{cost}(a_i, \lambda) = 1 \\ &\text{Insertion: } \text{cost}(\lambda, b_j) = 1 \\ &\text{Replacement: } \text{cost}(ai, bj) = 1 - SIM(a_i, b_j) \end{aligned} \tag{2}$$

In the literature, the dynamic-programming method [54] is commonly used to compute the edit distance. In the following, we use two event sequences S_1 = ⟨a,b,c⟩⟨b,d⟩⟨a,b,d⟩⟨e,f⟩⟨a,c⟩ and S_2 = ⟨a,b,c⟩⟨b,d⟩⟨a,c,d⟩⟨e,f⟩⟨a,e⟩ to show how it works. In Fig. 2, S_1 and S_2 are placed in the 1st column and the 1st row, respectively. The arithmetical series in the 2nd column and the 2nd row are filled as the initial values. The remaining cells in the matrix are then filled from the top-left corner to the bottom-right corner. Let S(i) be the prefix of S with i events and $C_{i,j}$ denote the cell corresponding to the ith event in S_1 and the jth event in S_2. The value of $C_{i,j}$ means the edit distance between $S_1(i)$ and $S_2(j)$. For instance, $C_{3,2}$ is filled with the edit distance between ⟨a,b,c⟩⟨b,d⟩⟨a,b,d⟩ and ⟨a,b,c⟩⟨b,d⟩, i.e., 1. Using the dynamic-programming method, the cell $C_{i,j}$ is filled with the minimum of three values, $C_{i,j-1} + \text{cost}(\lambda, b_j)$, $C_{i-1,j} + \text{cost}(a_i, \lambda)$, and $C_{i-1,j-1} + \text{cost}(a_i, b_j)$. For example, the value of $C_{5,3}$ (7/3) is obtained from three values, $C_{5,2}$ + cost(λ, ⟨a, c, d⟩), $C_{4,3}$+cost(⟨a,c⟩, λ), and $C_{4,2}$+cost(⟨a,c⟩, ⟨a,c,d⟩).

Since an SQ can be registered at any moment, we develop an algorithm that clusters the query n-grams one by one. Initially, the first query n-gram forms a new cluster. Let the query n-gram to be clustered be denoted as QNG. If there exist some clusters, for each cluster, the minimum and the maximum of the edit distances between QNG and the query n-grams in the same cluster are computed. The clusters whose maximum values do not exceed μ are then selected. If there is no such cluster, QNG itself forms a new cluster. Otherwise, among the selected clusters, QNG is assigned to the cluster whose minimum value is the smallest. After clustering, all the query n-grams in a cluster are stored as an inverted list.

Example 1 Let μ be set to 5/2. Consider the following query n-grams to be clustered.

- QNG_1 = ⟨a,b,c⟩⟨b,d⟩⟨a,b,d⟩⟨e,f⟩⟨a,c⟩
- QNG_2 = ⟨a,b,c⟩⟨b,d⟩⟨a,c,d⟩⟨e,f⟩⟨a,e⟩
- QNG_3 = ⟨a,b,c⟩⟨b,d⟩⟨a,d⟩⟨e,f⟩⟨a,e⟩
- QNG_4 = ⟨a,b,c⟩⟨b,d⟩⟨a,b,c,d⟩⟨e,f⟩⟨a,c⟩

- $QNG_5 = \langle a,b,c\rangle\langle e,f\rangle\langle a\rangle\langle e\rangle\langle c\rangle$
- $QNG_6 = \langle a,b,c\rangle\langle e,f\rangle\langle a,d\rangle\langle e\rangle\langle a,c\rangle$

Initially, a cluster C_1 is created to accommodate QNG_1. Since the edit distance between QNG_2 and QNG_1 is 7/6 ($<\mu$) as shown in Fig. 2, QNG_2 is assigned to C_1. For QNG_3, since edit-DIS(QNG_3, QNG_1)($= 1$) is larger than edit-DIS (QNG_3, QNG_2)($= 1/3$) and does not exceed μ, QNG_3 is also assigned to C_1. For QNG_4, because the maximum of the edit distances between it and all the query n-grams in C_1 is edit-DIS(QNG_4, QNG_3) (= 7/6), QNG_4 is assigned to C_1, too. However, QNG_5 itself forms a new cluster because the maximum of the distances between it and each query n-gram in C_1 is edit-DIS(QNG_5, QNG_3) (= 11/4), which is higher than μ. After that, the maximum of the edit distances between QNG_6 and all the query n-gram in C_1 is 2, while the maximum corresponding to C_2 is 1. QNG_6 is finally assigned to C_2 because the minimum between QNG_6 and C_2 (=1) is smaller than that between QNG_6 and C_1 (=11/6).

3.3 Cluster summarization

In addition to the query n-grams, for each cluster, two pieces of summary information are also recorded. One is a sequence of n events, called the *virtual n-gram*, where each event is the union of all the corresponding events in the query n-grams in the cluster. The other is the maximum of all the local error bounds of the query n-grams in the cluster, called the *global error bound*. Moreover, each event in a virtual n-gram is followed by the attributes *MAX* and *MIN*. For the ith event in the virtual n-gram, *MAX* (*MIN*) means the maximum (minimum) size of the ith events in all the query n-grams in the cluster.

The virtual n-gram, together with the MAX and MIN attributes, forms the *summarization* of the cluster. From the summarization, we can derive all the query n-grams that form the cluster, although some n-grams that do not belong to the cluster may also be mistaken. Pruning unmatched query n-grams in a cluster by computing the restricted edit distance between the data n-gram and the summarization is proved safe in Sect. 4.

Example 2 Suppose the local error bounds of QNG_1, QNG_2, QNG_3, and QNG_4 in Example 1 are 2, 2, 3, and 3, respectively. Since they constitute the cluster C_1, we produce the virtual n-gram VNG_1 for C_1 as shown in Fig. 3. VNG_1 is a sequence of five events, where the ith event in VNG_1 is the union of all the ith events in the four query n-grams. Therefore, the 1st event of VNG_1 is ⟨a,b,c⟩ with MAX 3 and MIN 3. Similarly, the 3rd event of VNG_1 is ⟨a,b,c,d⟩ with MAX 4 and MIN 2. As a result, VNG_1 is ⟨a,b,c⟩⟨b,d⟩⟨a,b,c,d⟩⟨e,f⟩⟨a,c,e⟩, where each event is followed by the corresponding MAX and MIN values.

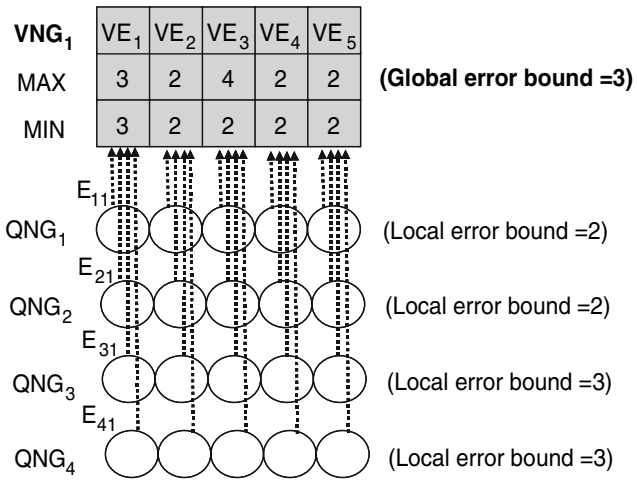

Fig. 3 The construction of a virtual n-gram (n = 5)

3.4 Query Buffer Creation

For each SQ composed of m query n-grams, we allocate m query buffers in the same order as that of the corresponding query n-grams in the SQ. The kth query buffer keeps the partial answers, obtained by merging the k approximate answers of the first k query n-grams of the SQ. The use of query buffers will be detailed in Sect. 5.

4 Pruning Mechanism

The pruning mechanism is designed to act like the indexing mechanism in the traditional DBMS in order to find all the approximate matches for each data n-gram. By utilizing the cluster summarization, the pruning mechanism computes the restricted edit distance between the data n-gram and each cluster. If it is above the global error bound, all the query n-grams in the cluster cannot be the approximate matches and are then pruned. In this way, the cost on distance computation is greatly reduced.

To compute the restricted edit distance as desired, a simplest approach is to compute the edit distance between the data n-gram and every n-gram that can be derived from the cluster summarization. Unfortunately, some n-grams that are not in the cluster may also be mistaken. Notice that in this case the resultant edit distance is still guaranteed less than or equal to the edit distance of every query n-gram in the cluster and can be set as the restricted edit distance. Furthermore, this simplest approach will also suffer from the heavy cost of distance computation. Therefore, we propose an original approach with much less computation cost to compute the restricted edit distance for each cluster.

For the ease of presentation, in the rest of this paper we denote a query n-gram in the cluster C, the virtual n-gram of C, and a data n-gram as QNG, VNG, and DNG, respectively. When computing the edit distance, the insertion and deletion

costs are fixed to 1, where as the replacement cost is variable according to the similarity between events. This motivates us to design a new cost function to substitute the replacement cost in Eq. (2) when computing the distance between DNG and VNG. For distinction, we denote the edit distance based on the new cost function as RE-DIS (DNG, VNG), which is used as the restricted edit distance.

Let the ith events in QNG and VNG be denoted as QE_i and VE_i, respectively. In addition, MAX_i and MIN_i are the summary information associated with VE_i. For each event in DNG, denoted DE, the replacement cost between it and VE_i, can be decided by using MAX_i and MIN_i. From Eq. (2), the new cost(DE, VE_i) for RE-DIS(DNG, VNG) should be the upper bound of SIM(DE, QE_i) for all QE_i in C. Therefore, we aim at finding a possible QE_i in C so that $|DE \cap QE_i|$ is maximized and $|DE \cup QE_i|$ is minimized, i.e., the value of $|DE \cap QE_i|/|DE \cup QE_i|$ is maximized. In the following, we present two observations and then consider three cases of the new cost function for replacement.

Observation 1 Maximize$|DE \cap QE_i|$

For each QE_i in C, since $QE_i \subseteq VE_i$, we have that $|DE \cap QE_i| \leq |DE \cap VE_i|$. Moreover, $|DE \cap QE_i| \leq MAX_i$ because $|QE_i| \leq MAX_i$. Therefore, for a possible QE_i in C, the upper bound of $|DE \cap QE_i|$ cannot be set to a value larger than the minimum of $|DE \cap VE_i|$ and MAX_i.

Observation 2 Minimize$|DE \cup QE_i|$

For each QE_i in C, since $DE \cup QE_i \supseteq DE$, we have that $|DE \cup QE_i| \geq |DE|$. Moreover, because $|QE_i| \geq MIN_i$ and $|DE \cup QE_i| = |DE| + |QE_i| - |DE \cap QE_i|$, we have $|DE \cup QE_i| \geq |DE| + MIN_i - |DE \cap QE_i|$, where the last term is the minimum of $|DE \cap VE_i|$ and MAX_i by Observation 1. Therefore, for a possible QE_i in C, the lower bound of $|DE \cup QE_i|$ cannot be set to a value smaller than the maximum of |DE| and $|DE| + MIN_i - \text{minimum}\{|DE \cap VE_i|, MAX_i\}$.

Based on the two observations, the new cost function can be defined as one of the following three cases.

Case 1: $MIN_i \leq |DE \cap VE_i| \leq MAX_i$

Since $|DE \cap VE_i| \leq MAX_i$, by Observation 1, we set the numerator to $|DE \cap VE_i|$. Moreover, since $MIN_i \leq |DE \cap VE_i|$, we have $|DE| + MIN_i - \text{minimum}\{|DE \cap VE_i|, MAX_i\} \leq |DE|$. By Observation 2, we set the denominator to |DE|. As a result, cost(DE, VE_i) is set to $1 - (|DE \cap VE_i|/|DE|)$.

Case 2: $|DE \cap VE_i| > MAX_i$

Since $|DE \cap VE_i| > MAX_i$, by Observation 1, we set the numerator to MAX_i. Moreover, since $|DE \cap VE_i| > MIN_i$, like Case 1, we set the denominator to |DE|. As a result, cost(DE, VE_i) is set to $1 - (MAX_i/|DE|)$.

Case 3: $|DE \cap VE_i| < MIN_i$

Since $|DE \cap VE_i| < MAX_i$, like Case 1, we set the numerator to $|DE \cap VE_i|$. Moreover, since $|DE \cap VE_i| < MIN_i$, we have $|DE| + MIN_i - \text{minimum}\{|DE \cap VE_i|, MAX_i\} > |DE|$. By Observation 2, we set the denominator to $|DE| + MIN_i - \text{minimum}\{|DE \cap VE_i|, MAX_i\}$, i.e., $|DE| + MIN_i - |DE \cap VE_i|$. As a result, cost(DE, VE_i) is set to $1 - (|DE \cap VE_i|/(|DE| + MIN_i - |DE \cap VE_i|))$.

To sum up, the replacement cost of RE-DIS(DNG, VNG) is formulated as follows:

$$\text{cost}(DE, VE_i) = \begin{cases} 1 - \frac{|DE \cap VE_i|}{|DE|}, & \text{if } MIN_i \leq |DE \cap VE_i| \leq MAX_i \\ 1 - \frac{MAX_i}{|DE|}, & \text{if } |DE \cap VE_i| > MAX_i \\ 1 - \frac{|DE \cap VE_i|}{|DE| + MIN_i - |DE \cap VE_i|}, & \text{if } |DE \cap VE_i| < MIN_i \end{cases} \quad (3)$$

Definition 2 (Restricted edit distance) The restricted edit distance between a data n-gram DNG and a virtual n-gram VNG, denoted as RE-DIS(DNG, VNG), is the minimum cost of the edit operations between the DNG and the n-grams derived from VNG. Let a_i and b_j be the ith event on DNG and the jth event on the n-gram derived from VNG, respectively. Let λ denote the null character. The edit operations and the corresponding costs are defined as follows:

Deletion: $\text{cost}(a_i, \lambda) = 1$

Insertion: $\text{cost}(\lambda, b_j) = 1$

Replacement: $\text{cost}(a_i, b_j)$ *based on Eq.* (3)

Recall the clustering of query n-grams in Sect. 3.2. For a query n-gram, we decide its cluster by computing its edit distance from every existing query n-gram. When the number of query n-grams increases, the clustering process can be time-consuming. An alternative way is to use the virtual n-grams and assign the query n-gram to the cluster with the minimum restricted edit distance. Similarly, if the minimum restricted edit distance exceeds μ, the query n-gram forms a new cluster. Although it is more efficient, the clusters can be looser. Therefore, we adopt the prior way of clustering in this paper.

We use Fig. 4 to illustrate the concept of restricted edit distance. Suppose each gray point in Fig. 4 is an n-gram and

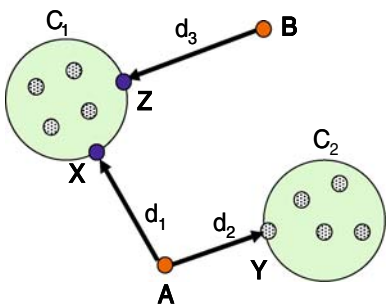

Fig. 4 The basic concept of restricted edit distance

there are two clusters C_1 and C_2. If data n-gram A arrives, we need an efficient way to check each cluster to see whether it can be pruned. For C_1, its minimum distance from A is d_1 and can be derived if we construct the black point X based on the virtual n-gram of C_1 and the associated MAX and MIN values. Although X may not be a real query n-gram in C_1, d_1 is guaranteed to be smaller than the distance between A and any query n-gram in C_1. In this way, our approach can make the decision by checking only the virtual n-gram instead of all the query n-grams in C_1. For C_1, we can construct Y in the same way although Y happens to be a real query n-gram. In fact, d_1 and d_2, respectively, stand for the restricted edit distances of C_1 and C_2 from A. If data n-gram B arrives, we may construct another point Z for C_1 to derive the restricted edit distance between B and C_1.

Based on Eq. (3), we derive two properties describing the relationship between the edit distance and the restricted edit distance. Property 1 says that the replacement cost between the data event and the virtual event is equal to or less than that between the data event and every query event contained in the virtual event. Moreover, Property 2 says that the restricted edit distance between the data n-gram and the virtual n-gram is equal to or less than the edit distance between the data n-gram and every query n-gram in the cluster to which the virtual n-gram belongs. Their proofs are shown respectively in the following.

Property 1 cost(DE, VE_i) *in* RE-DIS(DNG, VNG) $\leq$ cost(DE, QE_i) *in edit*-DIS(DNG, QNG), $\forall QE_i$ *of* QNG *in* C.

Proof Following Eq. (2) and Eq. (3) finds an upper bound of SIM(DE, QE_i), $\forall QE_i$ in C, which is to find the maximum of $|DE \cap QE_i|/|DE \cup QE_i|$ from each possible QE_i in C. For each of the three cases in Eq. (3), we prove the claim as follows:

Case 1: $MIN \leq |DE \cap VE_i| \leq MAX$

Since $QE_i \subseteq VE_i, \forall QE_i$ in $C \Rightarrow |DE \cap QE_i| \leq |DE \cap VE_i|, \forall QE_i$ in C; Moreover, $DE \cup QE_i \supseteq DE, \forall QE_i$ in $C \Rightarrow |DE \cup QE_i| \geq |DE|, \forall QE_i$ in C; Thus, $1 - (|DE \cap VE_i|/|DE|) \leq 1 - (|DE \cap VE_i|/|DE \cup QE_i|) \leq 1 - (|DE \cap QE_i|/|DE \cup QE_i|) = 1 - SIM(DE, QE_i), \forall QE_i$ in C.

Case 2: $|DE \cap VE_i| > MAX$

Since $|QE_i| \leq MAX, \forall QE_i$ in $C \Rightarrow |DE \cap QE_i| \leq MAX < |DE \cap VE_i|, \forall QE_i$ in C; like Case 1, $|DE \cup QE_i| \geq |DE|, \forall QE_i$ in C; Therefore, $1-(MAX/|DE|) \leq 1-(MAX/|DE \cup QE_i|) \leq 1 - (|DE \cap QE_i|/|DE \cup QE_i|) = 1 - SIM(DE, QE_i), \forall QE_i$ in C.

Case 3: $|DE \cap VE_i| < MIN$

Since $|DE \cap VE_i| < MAX \Rightarrow$ like Case 1, $|DE \cap QE_i| \leq |DE \cap VE_i|, \forall QE_i$ in C; Moreover, $|QE_i| \geq MIN, \forall QE_i$ in $C \Rightarrow |DE \cup QE_i| = |DE| + |QE_i| - |DE \cap QE_i| \geq |DE| + MIN - |DE \cap QE_i| \geq |DE| + MIN - |DE \cap VE_i|, \forall QE_i$ in C; Therefore, $1 - (|DE \cap VE_i|/(|DE| + MIN - |DE \cap VE_i|)) \leq 1 - (|DE \cap VE_i|/|DE \cup QE_i|) \leq 1 - (|DE \cap QE_i|/|DE \cup QE_i|) = 1 - SIM(DE, QE_i), \forall QE_i$ in C.

Property 2 RE-DIS(DNG, VNG) $\leq$ *edit*-DIS(DNG, QNG), $\forall$QNG *in C.*

Proof Following Definition 1, $\forall$QNG in C, we can find a sequence of edit operators $O = \langle O_1 O_2, \ldots, O_n \rangle$ that has the minimal cost, i.e., edit-DIS(DNG, QNG), to transform DNG into QNG. Let D_O be the cost for transforming DNG into VNG by using O and the cost function in Eq. (3). Among the operators in O, the insertion and deletion costs in D_O are the same as those in edit-DIS(DNG, QNG). Owing to Property 1, the replacement cost is smaller in D_O. Therefore, $D_O \leq$ edit-DIS(DNG, QNG), $\forall$QNG in C. Since we compute RE-DIS(DNG, VNG) in either of the following two cases, we prove the claim for each case:

Case 1: If O incurs the minimal cost in transforming DNG into VNG, RE-DIS(DNG, VNG) $= D_O \leq$ edit-DIS(DNG, QNG), $\forall$QNG in C.

Case 2: If using O to transform DNG into VNG does not incur the minimum cost, there must exist a sequent of edit operators so that RE-DIS(DNG, VNG) $< D_O \leq$ edit-DIS(DNG, QNG), $\forall$QNG in C.

Example 3 Assuming that VNG_1 in Example 2 is associated with a series of [MIN,MAX] pairs, [3,3], [2,2], [2,4], [2,2], and [2,2]. Given a data n-gram DNG = ⟨a,b,c⟩⟨b,d⟩⟨a,b,d⟩⟨e,f⟩⟨a,c,e⟩, the matrix for computing RE-DIS(DNG, VNG_1) is shown in Fig. 5. Note that the replacement cost in Eq. (3) is used. For instance, in $C_{3,3}$, Case 1 is applied because $|\langle a,b,d \rangle \cap \langle a,b,c,d \rangle|$ is in the range [2,4]. Moreover, in $C_{5,5}$, Case 2 is applied because $|\langle a,c,e \rangle \cap \langle a,c,e \rangle| = 3$ is larger than $MAX_5 = 2$. It verifies that the restricted edit distance (=1/3) will not exceed the edit distance computed from the query n-gram QNG_2 (=5/6), which can be computed by the method described in Sect. 3.2.

If each SQ is composed of only one query n-gram, pruning a cluster of query n-grams by the restricted edit distance is safe because no answer will be lost due to Property 2. However, for the SQ composed of more than one query

DNG \ VNG_1		<a,b,c>	<b,d>	<a,b,c,d>	<e,f>	<a,c,e>
	0	1	2	3	4	5
<a,b,c>	1	0	1	2	3	4
<b,d>	2	1	0	1	2	3
<a,b,d>	3	2	1	0	1	2
<e,f>	4	3	2	1	0	1
<a,c,e>	5	4	3	2	1	1/3

Fig. 5 The RE-DIS(DNG, VNG_1) matrix

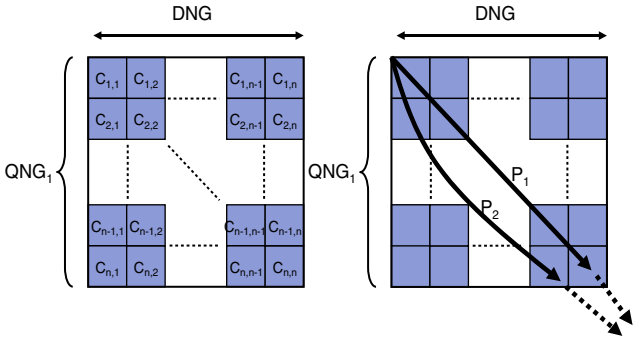

Fig. 6 Two paths crossing the matrix for computing edit-DIS(DNG, QNG_1)

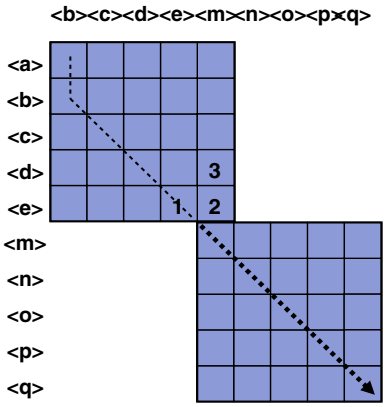

Fig. 7 A path crossing two matrices

n-gram, it is unsafe to prune a cluster of query n-grams just because the restricted edit distance exceeds the global error bound. The reason can be seen from the computation of the edit distance between a data n-gram and a query n-gram. Suppose that an SQ is composed of two query n-grams QNG_1 and QNG_2. At the left-hand side of Fig. 6, the value of $C_{n,n}$ in the matrix indicates edit-DIS(DNG, QNG_1). At the right-hand side of Fig. 6, we assume that P_1 is a path with the minimum cost from the top-left corner to the bottom-right corner in this matrix. Although the value of $C_{n,n}$ in P_1 exceeds the error bound of the SQ, it is unsafe to prune QNG_1. Let FA and P, respectively, denote a final answer of SQ that includes DNG and a path with the minimum cost from the top-left corner to the bottom-right corner in the matrix for computing edit-DIS(FA, SQ). If P contains P_2 instead of P_1, FA will be lost. As a result, even though the value of $C_{n,n}$ exceeds the error bound, DNG still has a chance to be a partial answer of the SQ when the value of $C_{n,n-1}$ does not exceed the error bound. Therefore, we have to further check some cells in the matrix for computing the edit distance and select the minimum as the lower bound for safely pruning a query n-gram.

Example 4 For simplicity, each event in this example has only one item. Let SQ consist of $QNG_1 = \langle a\rangle\langle b\rangle\langle c\rangle\langle d\rangle\langle e\rangle$ and $QNG_2 = \langle m\rangle\langle n\rangle\langle o\rangle\langle p\rangle\langle q\rangle$. The error bound is *1*. A data segment D is $\langle b\rangle\langle c\rangle\langle d\rangle\langle e\rangle\langle m\rangle\langle n\rangle\langle o\rangle\langle p\rangle\langle q\rangle$, where DNG_1 and DNG_5 are $\langle b\rangle\langle c\rangle\langle d\rangle\langle e\rangle\langle m\rangle$ and $\langle m\rangle\langle n\rangle\langle o\rangle\langle p\rangle\langle q\rangle$, respectively. In this example, we skip discussing the processing of DNG_2, DNG_3, and DNG_4 for brevity. The matrices for computing edit-DIS(DNG_1, QNG_1) and edit-DIS(DNG_5, QNG_2) are illustrated in Fig. 7, respectively. In the first matrix, if we only consider the value in $C_{5,5}$, DNG_1 will not form a partial answer of the SQ because the value in $C_{5,5}$ (=2) exceeds the error bound. However, if we skip DNG_1 at this moment, we will eventually miss D in the final answers. Therefore, we take the minimum among the three cells $C_{4,5}$, $C_{5,5}$ and $C_{5,4}$ as the criterion to be checked. Since in this case the minimum (1) does not exceed the error bound, DNG_1 will be kept in the query buffer of QNG_1, and the final answer D can be identified as we merge DNG_1 with DNG_5.

Since the restricted edit distance is a lower bound of the edit distance between the data n-gram and each query n-gram in the cluster, the above scenario can also be applied to the matrix for computing the restricted edit distance for each cluster. The cells that we have to check, including cell $C_{n,n}$, are named the *acceptable cells*, and can be located via a formula as follows. Let the global error bound be denoted as δ. Since the path of a final answer must cross the boundaries at the right or the bottom of the matrix, the acceptable cells must be a subset of the cells in the last row or the last column of the matrix. Among these cells, all the cells $C_{i,j}$, where $i < n-\delta$ or $j < n-\delta$, cannot be the acceptable cells because any path crossing them must incur the insertion or deletion cost higher than δ. Namely, only the remaining cells should be considered as the acceptable cells. As a result, the cluster of query n-grams can be safely pruned only if all the values in these acceptable cells exceed the global error bound. Therefore, we choose the minimum value in acceptable cells as the *global distance*, denoted as *global*-DIS(DNG, VNG).

$$
\begin{aligned}
&\textit{global}\text{- DIS(DNG,VNG)}\\
&= \text{MINIMUM}\{V(C_{i,j})|C_{i,j} \text{ belongs to } M\text{(DNG,VNG)},\\
&(i = n, n-\delta \le j \le n) \vee (j = n, n-\delta \le i < n)\}\\
&\text{where } M\text{(DNG, VNG) is the matrix for computing}\\
&\qquad \text{RE-DIS(DNG, VNG)}\\
&\qquad \delta \text{ is the global error bound}\\
&\qquad V(C_{i,j}) \text{ is the value of } C_{i,j}
\end{aligned}
\tag{4}
$$

Following Eq. (4), we derive Lemma 1, saying that the global distance between the data n-gram and the virtual n-gram must be equal to or less than the distance between the data n-gram and every query n-gram in the cluster to which the virtual n-gram belongs. Therefore, no answer will be lost and the correctness of the pruning mechanism is guaranteed.

Lemma 1 $\forall$QNG *in C, global*-DIS(DNG, VNG) $\le$ *edit*-DIS(DNG, QNG).

Proof Following Eq. (4), it is obvious that global-DIS (DNG,VNG) $\leq$ RE-DIS(DNG,VNG) and then by Property 2 it leads to the claim.

5 Merging mechanism

Whenever a query n-gram is identified as an approximate match of the incoming data n-gram, the pruning mechanism sends a copy of the data n-gram to the corresponding query buffer in the merging mechanism. The query buffers of each SQ aim at collecting all the data n-grams received from the pruning mechanism to form the candidate answers of the SQ. Since a candidate answer of the SQ can be composed of multiple data n-grams that arrive at different time, the merging mechanism is designed in such a way that the data n-grams can be incrementally merged and kept as the partial answers. Moreover, a data n-gram can be an approximate answer of more than one query n-gram of the same SQ. Therefore, the merging mechanism may use a data n-gram to grow different partial answers of the same SQ simultaneously.

In our approach, the length of the n-gram, n, is set to be larger than the error bound of any SQ. It follows that any segment on the event stream, which does not contain an approximate match of any query n-gram in the SQ, is impossible to be an answer of the SQ. Therefore, only the candidate answers generated from query buffers must be examined. For the SQ composed of m query n-grams, there will be m query buffers, which are numbered with the order of query n-grams in the SQ. The kth query buffer keeps every partial answer that merges the k approximate answers corresponding to the first k query n-grams of the SQ.

When a data n-gram is sent to the kth query buffer of an SQ, four criteria are used in series to check whether it can be merged with any partial answer in the $(k-1)$th query buffer to form a new partial answer in the kth query buffer. The common idea behind the criteria is that the *estimated* lower bound of the edit distance between the new partial answer and the SQ cannot exceed the error bound of the SQ. Moreover, the computation results for the kth query buffer can be reused to estimate the lower bounds for the partial answers in the $(k+1)$th query buffer. In this way, the costs on distance computation and the estimation of the lower bound are greatly reduced. In the following, we introduce the five steps of how a query buffer deals with a data n-gram. The first four steps correspond to the four criteria, respectively, and the last step is to handle the new partial answers generated. For the ease of presentation, we denote the k-th query n-gram in an SQ as QNG_k and the prefix of SQ with the first k query n-grams as $\text{SQ}_{1..k}$. Without loss of generality, we assume that the SQ associated with an error bound ε has m query buffers and the incoming data n-gram, denoted as DNG, is sent to the k-th query buffer, denoted as QB_k. The treatments for the initialization and completion in the first and the last query buffers (QB_1 and QB_m) will also be noted in the subsequent discussions. In addition, the partial answer that currently exists in QB_{k-1} is denoted as PA_{k-1}, where as the new answer that merges DNG and PA_{k-1} is denoted as $\text{PA}_{k-1} \oplus \text{DNG}$.

Step 1: Existence of PA_{k-1}

Since the query buffers follow the order of query n-grams in the SQ, it can be seen that every partial answer in QB_k must have a corresponding partial answer in QB_{k-1} and the latter always appears earlier than the former. Therefore, this step is to check whether there is any partial answer in QB_{k-1}. If not, DNG is discarded and no partial answer is generated. Note that this check is skipped when $k = 1$.

Step 2: Lengths of $\text{PA}_{k-1} \oplus$ DNG and $\text{SQ}_{1..k}$

As afore mentioned, the difference between the length of the SQ and that of its answer implies a lower bound of their edit distance. Therefore, for each partial answer in QB_{k-1}, we merge it with DNG and compare the length of the merged result with that of $\text{SQ}_{1..k}$, i.e. $n \times k$. If this difference exceeds ε, this partial answer is ignored because it will never be a part of any approximate answer of the SQ. Note that this check is also skipped when $k = 1$.

Step 3: Distance between DNG and QNG_k

For the same reason we discussed in Sect. 4, it is unsafe to ignore DNG just because the edit distance between DNG and QNG_k exceeds ε. Instead, we have to check the acceptable cells, as defined in Eq. (4), in the matrix for computing edit-DIS(DNG, QNG_k) and select the minimum as the estimate, which is called the *local distance* and denoted as local-DIS(DNG, QNG_k). If this estimate exceeds the error bound, DNG is discarded and no partial answer is generated. Note that this step is executed for any k. We provide the formula for computing the local distance as follows:

$$\begin{aligned}&\text{local -DIS(DNG,QNG}_k\text{)}\\&= \text{MINIMUM}\{V(C_{i,j}) | C_{i,j} \text{ belongs to } M(\text{DNG,QNG}_k),\\&(i = n, n-\varepsilon \leq j \leq n) \vee (j = n, n-\varepsilon \leq i < n)\}\end{aligned} \quad (5)$$

where M(DNG,QNG_k) is the matrix for computing edit-DIS (DNG, QNG_k)

ε is the error bound

$V(C_{i,j})$is the value of$C_{i,j}$

Following Eq. (5), we derive Property 3, saying that the local distance between the data n-gram and the query n-gram must be equal to or less than the edit distance between them.

Property 3 *local*-DIS(DNG, QNG) $\leq$ *edit*-DIS(DNG, QNG), $\forall$QNG.

Proof Following Eq. (5), it is obvious that this claim is true, because the cell $C_{\text{n,n}}$ in the edit-DIS(DNG, QNG) matrix is always included in the acceptable cells for computing local-DIS(DNG, QNG).

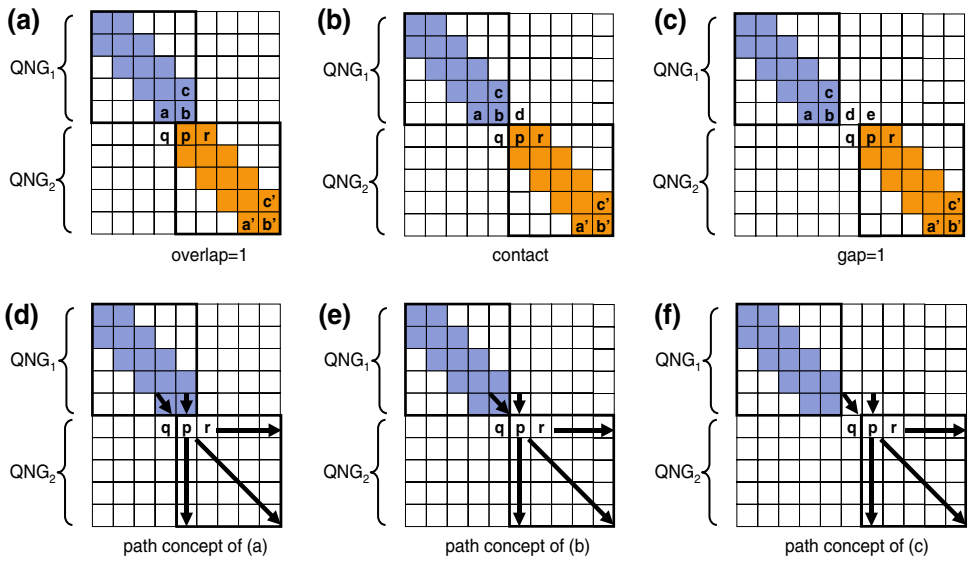

Fig. 8 Computation of the partial distance

Step 4: Distance between $\mathbf{PA}_{k-1} \oplus \mathbf{DNG}$ and $\mathbf{SQ}_{1..k}$

Like Steps 1 and 2, this check is skipped when $k = 1$. For the ease of presentation, we use an example for $k = 2$ as shown in Fig. 8 to explain this check. The error bound is set to *1* and therefore each matrix only has three acceptable cells. Assume that the data n-gram DNG_1 has been kept as a partial answer in the query buffer of QNG_1, and the values in the cells a, b, c are recorded. If the data n-gram DNG is sent to the query buffer of QNG_2, by the previous step we will obtain local-DIS(DNG, QNG_2). As shown in Fig. 8a–c, there are three types of temporal relationships between DNG_1 and DNG, including *overlap*, *contact*, and *gap*. For each type, with the same idea of the acceptable cells, we estimate the lower bound of the minimum among the three cells a', b', c'. If this estimate exceeds the error bound, this merged result ($\text{DNG}_1 \oplus \text{DNG}$) is ignored because it cannot be a part of any approximate answer of the SQ. We call such an estimate the *partial distance* and denote it as $part - \text{DIS}(\text{DNG}_1 \oplus \text{DNG}, \text{SQ}_{1..2})$. For efficiency, we propose a method to compute the partial distance using only the values of the cells a, b, c and local-DIS(DNG, QNG_2).

Since the error bound is *1*, only the cells colored are involved in the estimation of the partial distance. The goal is to find a path with the minimum cost from the top-left corner to either of the cells a', b', c'. It can be seen that this path must pass cell p or cell q. Moreover, since the data n-grams can be partially overlapped, the computation can be redundant. For instance, a path that passes cell p and ends at cell b' in Fig. 8a is equivalent to the path that passes cell q and ends at cell a' in Fig. 8b. To avoid the redundancy, we propose the *path concept* as indicated in Fig. 8d–f to limit the paths considered at each check to only the ones passing cell p, i.e., the cell $C_{1,1}$ in the *edit-DIS(DNG*, QNG_2) matrix. If the path with the minimum cost is the one passing cell q, it still can be found at later checks. In the following, we take Fig. 8 as a running example to show how to compute the partial distance for each type.

Type A: Overlap

As Fig. 8(a) depicts, DNG overlaps with DNG_1. By Eq. (5), local-DIS(DNG, QNG_2) means the minimum cost of a path from cell p to either of the cells a', b' and c'. Moreover, a path passing cell p must pass either cell a or cell b, while the value in each of these cells stands for the minimum cost of a path from the top-left corner to it. Since the minimum cost of a path from cell $a(b)$ to cell p is *0* (*1*, the deletion cost), the partial distance can be formulated as:

$$\begin{aligned}&\text{part-DIS}(\text{DNG}_1 \oplus \text{DNG}, \text{SQ}_{1..2})\\ &= \text{minimum}\{\text{V}(a), \text{V}(b)+1\} + \text{local-DIS}(\text{DNG}, \text{QNG}_2)\end{aligned} \tag{6}$$

Note that if the number of overlaps exceeds the error bound, this merged result can be ignored directly.

Type B: Contact

As shown in Fig. 8b, cell p is the contact of the two matrices. Unlike the previous type, a path passing cell p must pass either cell b or cell d, while the value of cell d is unknown. Fortunately, we can compute a lower bound of it from the cells c and b because a path passing cell d must pass either cell b or cell c. The minimum cost of a path from cell $c(b)$ to cell d is *0* (*1*, the insertion cost), and thus the value of cell d is set to minimum$\{\text{V}(c), \text{V}(b)+1\}$. Since the minimum cost of a path from cell b (d) to cell p is *0* (*1*, the deletion cost), the partial distance can be formulated as:

$$\begin{aligned}&\text{part-DIS}(\text{DNG}_1 \oplus \text{DNG}, \text{SQ}_{1..2})\\ &= \text{minimum}\{\text{V}(b), \text{V}(d)+1\} + \text{local-DIS}(\text{DNG}, \text{QNG}_2)\end{aligned} \tag{7}$$

Type C: Gap

As shown in Fig. 8c, there is a gap between the arrival times of DNG and DNG_1. A path passing cell p must pass

either cell d or cell e, where the values of both cells are unknown. Following the previous type, we can compute both lower bounds of the value of cell d, minimum{V(c), V(b)+ 1}, and the value of cell e, minimum{V(c)+ *1*, V(d)+ 1}. Since the minimum cost of a path from cell d (e) to cell p is *0* (*1*, the deletion cost), the partial distance can be formulated as:

$$\begin{aligned}&\text{part-DIS}(\text{DNG}_1 \oplus \text{DNG}, \text{SQ}_{1..2})\\ &= \text{minimum}\{\text{V}(d), \text{V}(e) + 1\} + \text{local-DIS}(\text{DNG}, \text{QNG}_2)\end{aligned} \tag{8}$$

Note that if the number of gaps exceeds the error bound, this merged result can be ignored directly, and the partial answer DNG_1 is also removed. The reason for the latter is that the gap between DNG_1 and any of the subsequent data n-grams will be getting larger, and no partial answer can be generated in the future. Notice that Eqs. (6) and (8) can be easily generalized to the cases with their overlaps or gaps larger than 1.

Following Eqs. (6), (7), and (8), we get Lemma 2. It says that, for any query and $k > 1$, we need not merge the incoming data n-gram, which is the approximate answer of QNG_k, with a partial answer in QB_{k-1} if the partial distance between the new partial answer ($\text{PA}_{k-1} \oplus \text{DNG}$) and the prefix of SQ with the first k query n-grams ($\text{SQ}_{1..k}$) exceeds the error bound of the query.

Lemma 2 *In each type, it is safe to ignore $PA_{k-1} \oplus DNG$ if part-DIS($PA_{k-1} \oplus DNG$, $SQ_{1..k}$) $> \varepsilon$, $\forall PA_{k-1}$ in QB_{k-1}.*

Proof Without loss of generality, we use the example in Fig. 8 again. In this example, we set k = 2 and $\varepsilon = 1$. In each type, the paths considered at each check are limited to the ones passing cell p. Since the path with the minimum cost can either pass cell p or not, we prove the claim for each case as follows:

Case 1 If the path as desired passes cell p, according to Property 3, part-DIS($\text{DNG}_1 \oplus \text{DNG}$, $\text{SQ}_{1..2}$), must be smaller than or equal to edit-DIS($\text{DNG}_1 \oplus \text{DNG}$, $\text{SQ}_{1..2}$). Therefore, if the partial distance exceeds ε, it is safe to ignore $\text{DNG}_1 \oplus \text{DNG}$ because merging $\text{DNG}_1 \oplus \text{DNG}$ with subsequent data n-grams will not result in a final answer of the SQ.

Case 2 If the path as desired does not pass cell p, it must exist in one of the latter checks. Take Fig. 8a as an example. Suppose that the path with the minimum cost passes cell r instead of cell p and this path ends at cell b′ in Fig. 8a. It is safe to ignore $\text{DNG}_1 \oplus \text{DNG}$ because this merged result, i.e., the value of cell a' in Fig. 8b, will be taken back at a latter check.

As aforementioned, the first four steps are used in series to check whether an incoming data n-gram can be merged with a previous partial answer. We decide the order of the four steps by making a compromise between the pruning effect and the computation cost. Step 1 is undoubtedly the fastest step because only one query buffer should be checked. However, if we only use Step 1 for pruning, there will be many unnecessary candidates retained for the final check, e.g., too long candidates. The pruning effect is improved if Steps 2 and 3 are also included. Compared with the length difference for Step 2, the distance computation by dynamic programming in Step 3 is obviously more time-consuming. This distinction leads to the current order of the two steps. Step 4 is the most complicated out of the four, because Since it needs the computation results of Step 3 to estimate the required distances, we let it be the last step for pruning.

Step 5: Generation of $\text{PA}_{k-1} \oplus \text{DNG}$ in QB_k

After the four checks are passed, a partial answer, $\text{PA}_{k-1} \oplus \text{DNG}$, is generated and stored in QB_k. For $k = 1$, this step is to directly store DNG as a partial answer in QB_k. For $k = m$, this step is not to store anything into the query buffer. Instead, a candidate answer is produced, and the edit distance between it and the SQ is computed to determine whether it can be a final answer. Moreover, from the matrix for computing this edit distance, we also check every prefix of the candidate answer with lengths in $[L - \varepsilon, L - 1]$, where L is the length of the candidate answer, to see whether it is a final answer. In this way, more than one final answer can be obtained via only one computation of the edit distance. Note that redundant answers produced at different time can be identified with a simple check.

To guarantee that no approximate answer of a query will be lost, we provide Theorem 1 as follows:

Theorem 1 *For each SQ, all the final answers on the event stream are included in the set of candidate answers produced by our approach.*

Proof For an SQ with the error bound ε and length L, only the data segments with lengths in $[L-\varepsilon, L+ \varepsilon]$ on the event stream can be an answer. Monitoring all the data segments with lengths in $[L-\varepsilon, L+ \varepsilon]$ on the event stream can ensure the claim because all possible answers are included in the set of candidate answers. From this viewpoint, the two mechanisms in our approach act like filters to reduce the number of candidate answers by eliminating as early as possible the ones that cannot be final answers. Therefore, what we need to prove is that using our mechanisms will not result in false elimination of any final answer. Assume that a final answer FA of the SQ is not in the set of candidate answers. It implies that a data n-gram DNG in FA, corresponding to the query n-gram QNG_k in the SQ, is falsely pruned by one of the two mechanisms.

Case 1 **Pruning mechanism**

In the pruning mechanism, DNG is discarded only if the cluster containing QNG_k is pruned. In this situation, from Lemma 1, edit-DIS(DNG, QNG_k) exceeds ε and therefore edit-DIS(FA, SQ) also exceeds ε. This leads to a contradiction.

Case 2 **Merging mechanism**

In the merging mechanism, DNG is discarded only if for each partial answer PA_{k-1} in QB_{k-1} of the SQ, which is contained in FA, $PA_{k-1} \oplus DNG$ is pruned due to either of the criteria used in the first four steps.

Step 1: In this step, DNG is discarded directly if there is no partial answer in QB_{k-1} of the SQ. It also means that no data segment on the event stream can be merged with DNG to form an approximate answer of $SQ_{1..k}$. This leads to the same contradiction in Case 1.

Step 2: In this step, $PA_{k-1} \oplus DNG$ is pruned only if the difference between its length and $SQ_{1..k}$ exceeds ε. It implies that edit-DIS(FA, SQ) must be larger than ε and therefore leads to the same contradiction in Case 1.

Step 3: In this step, DNG is discarded directly if local-DIS(DNG, QNG_k) exceeds ε. By Property 3, edit-DIS(DNG, QNG_k) is also larger than ε. Therefore, the same contradiction in Case 1 can be drawn.

Step 4: In this step, $PA_{k-1} \oplus DNG$ is pruned only if part-DIS($PA_{k-1} \oplus DNG$, $SQ_{1..k}$) exceeds ε. By Lemma 2, there are two cases to consider. For Case 1, edit-DIS($DNG_1 \oplus DNG$, $SQ_{1..k}$) is larger than ε and therefore edit-DIS(FA,SQ) will also large than ε. This leads to a contradiction. For Case 2, we can still obtain FA by merging PA_{k-1} with the subsequent data n-gram to fit Case 1. According to the proofs in Steps 1, 2, and 3, such a data n-gram will not be pruned. This also leads to a contradiction.

6 Experimental evaluation

In this section, a series of experiments are performed to demonstrate the efficiency as well as the scalability of our approach compared with a naïve approach. In addition, we also illustrate how the two parameters, the length of n-gram (n) and the clustering threshold (μ), influence the performance of our approach. Since we consider the symbolic music data in the stream environment, no distortion will occur in the concerned music stream. Moreover, we exclude the noises that may arise during the data transmission from the sender to the receiver because such noise hardly exists in the symbolic music stream in the real situation. According to the problem definition, all the music segments within the error tolerance of a range query are regarded as the desired answers. In the experiments, the correctness of our approach is guaranteed because all the music segments within the given error tolerance of an SQ are derived.

6.1 Experiment setting

We implement a prototype system to construct a streaming environment with multiple music streams. To simulate a music stream, we randomly play 100 songs in the MIDI format as the testing data. The genres of the 100 songs include Blues, Classical, Country, Folk, Jazz, and etc. The total time to play all the 100 songs is about 2,6805 s whereas the total number of events is 168,283. Note that music objects in the format of SMR, such as MusicXML [41], are also allowable. We also implement a translator in the system to continuously receive the music objects and transform each SMR into the pitch representation to form the corresponding event stream. Since it takes less than a second to obtain the pitch representation from each SMR, this transformation time is not included in the experiment results.

The user can issue any musical segments as the SQs via the interface of the system. Thereafter, the system continuously monitors the music stream and notifies the user when songs on the music stream contain a segment close to the issued SQ. To evaluate our approach, all the SQs in the experiments are extracted from the 100 songs. There are two steps of query generation in our experiments. In the first step, each SQ is a music segment with length 24 randomly selected from the 100 songs. If the SQ is selected from the refrain part or repeating phrases of a song, we may likely find the approximate answers (i.e., the music segments within the error tolerance of the SQ) in the same song. Moreover, the approximate answers may also occur in other songs. In the second step, each SQ is slightly adjusted according to the corresponding error bound ε as follows. We first randomly select ε events from the selected SQ, and then remove a symbol from each of the selected events to generate a new SQ for the experiments. If the length of the newly generated SQ is less than 24, we concatenate the event(s) behind the original SQ with the adjusted SQ to fulfill the length requirement. The length of n-gram (n) and the clustering threshold (μ) influence the performance of our approach and will be discussed in Sects. 6.4 and 6.5, respectively.

A naïve approach is implemented as a benchmark for evaluating the performance of our approach. Since the length of each SQ is 24 and their error bounds (ε) are equal, the lengths of the final answers must be in $[24-\varepsilon, 24+\varepsilon]$. The naïve approach uses a sliding window with length $24+\varepsilon$ on the event stream and continuously extracts the most recent events to form a candidate answer. For each candidate answer, the edit distance between it and each SQ is calculated with the cost functions in Definition 1. Note that for each candidate answer and each SQ, only one computation is sufficient to determine whether the candidate answer and its prefixes with lengths in $[24-\varepsilon, 23+\varepsilon]$ can be the final answers of the corresponding SQ. The two approaches are implemented in C++ and run on the PC with Pentium 4 CPU of 2.80 GHz and 512 MB memory.

6.2 Experiments on real-time requirement

Since our approach is operated in the streaming environment, the most important criterion on efficiency is to demonstrate

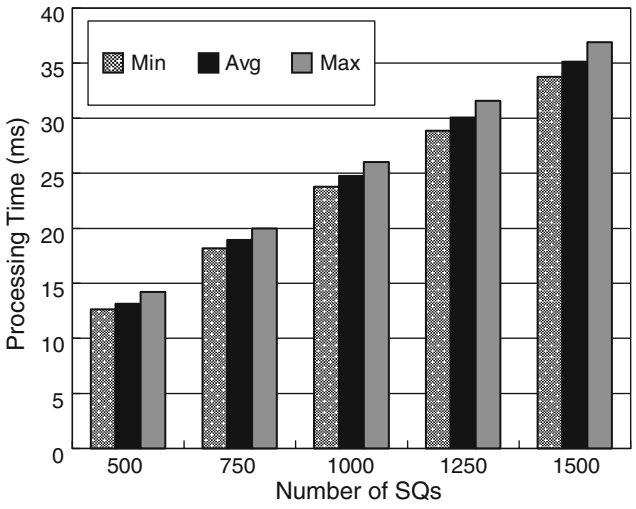

Fig. 9 The processing time for different numbers of SQs by our approach

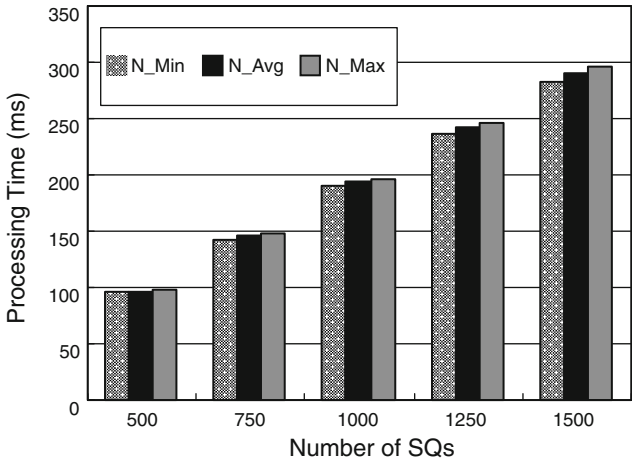

Fig. 10 The processing time for different numbers of SQs by the naïve approach

whether our approach can meet the real-time requirement. In our system, a new data n-gram will be generated when a new event arrives. Based on the system architecture shown in Fig. 1, an incoming data n-gram will trigger both the pruning mechanism and the merging mechanism at once. Therefore, we measure the processing time of a data n-gram when a new event arrives.

In the first experiment, we try different numbers of SQs from 500, 750, 1,000, 1,250, to 1,500. The error bound of each SQ is set to 2. In addition, the parameter n is set to 6, and in this case each SQ is decomposed into 4 query n-grams. We measure the average processing time (Avg), the minimum processing time (Min), and the maximum processing time (Max) of our approach when a new event arrives.

The results are depicted as a histogram in Fig. 9, where the x-axis indicates the number of SQs and the y-axis indicates the processing time. Under the same condition, the average processing time (N_Avg), the maximum processing time (N_Max), and the minimum processing time (N_Min) of the naïve approach are measured and shown in Fig. 10. Our approach outperforms the naïve approach significantly, especially as the number of SQs increases. The excessive processing time of the naïve approach results from two parts. First, the computation for different SQs is not shared. Second, the naïve approach spends too much time on processing the data segments that are impossible to be the approximate answers. Therefore, the N_Avg grows linearly with the number of SQs. On the contrary, we develop several techniques to significantly reduce the processing time, including sharing the computation among similar query n-grams, pruning the unmatched data n-grams as soon as possible, and reusing the intermediate results as much as possible. In the real-time environment, a realistic approach should be able to process an incoming data n-gram in time without any delay. In the above experiments, a new event arrives at the system for every 159 ms on average. The results in Fig. 10 reveal that the naïve approach will not be able to process a data n-gram in time even for only 1,000 SQs. However, our approach takes only 24 ms on average under the same condition. Therefore, based on the discussion above, the processing time of our approach is reasonable and practical for real-time requirement.

To study the case when the length of each SQ may be different, we produce 500 SQs with lengths 12, 18, and 24, respectively. The setting of other parameters is the same as the above experiments. The Avg is about 27.23 ms while the N_Avg is about 263.46 ms. Our approach still outperforms the naïve approach in this situation. Compared with the Avg of the previous experiment for 1,500 SQs (see Fig. 9), the Avg of this experiment is shorter. The number of query n-grams in this experiment is 4,500 whereas the number of query n-grams in the previous experiment for 1,500 SQs is 6,000. Since the two experiments use the same clustering threshold, the number of clusters generated in this experiment is no doubt smaller. It follows that the number of clusters to be examined in the pruning mechanism is also reduced in this experiment. It is known that to generate the final answer of an SQ with longer length, more partial answers have to be kept and checked. Accordingly, the number of partial answers to be checked in the merging mechanism is also reduced in this experiment.

6.3 Experiments on scalability

To demonstrate the scalability of our approach, we compare the total processing time of the two approaches for different numbers of SQs and the results are shown in Fig. 11. Note that the horizontal line stands for the total playing time of the 100 songs, which is about 7.5 h. As Fig. 11 indicates, the total processing time of the naive approach increases dramatically

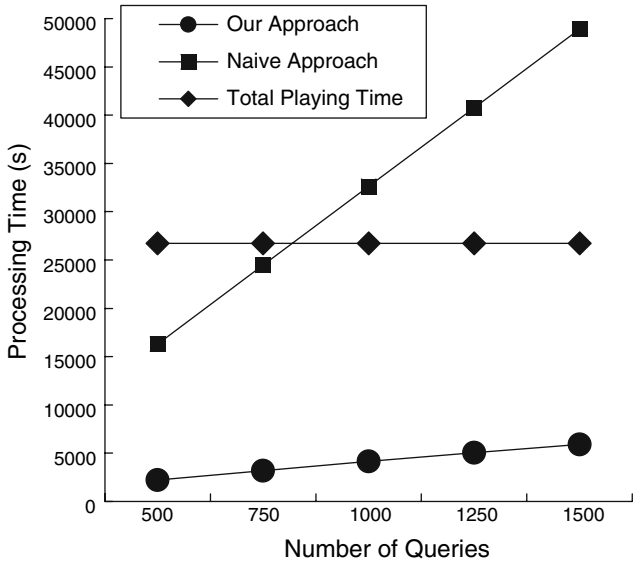

Fig. 11 Total processing time for different numbers of SQs on one music stream

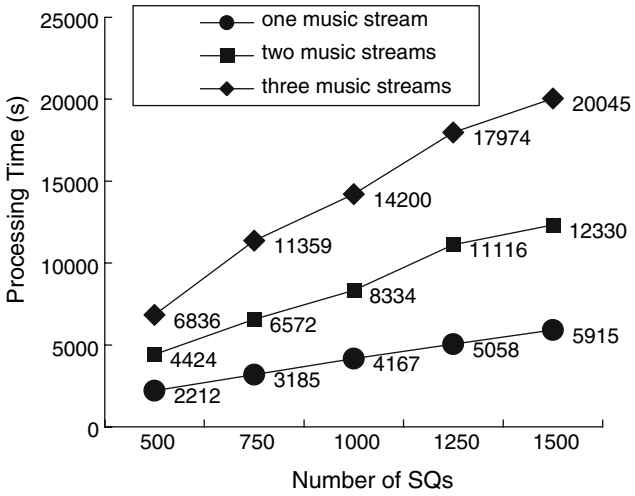

Fig. 12 Total processing time of different numbers of SQs for one, two, and three music streams

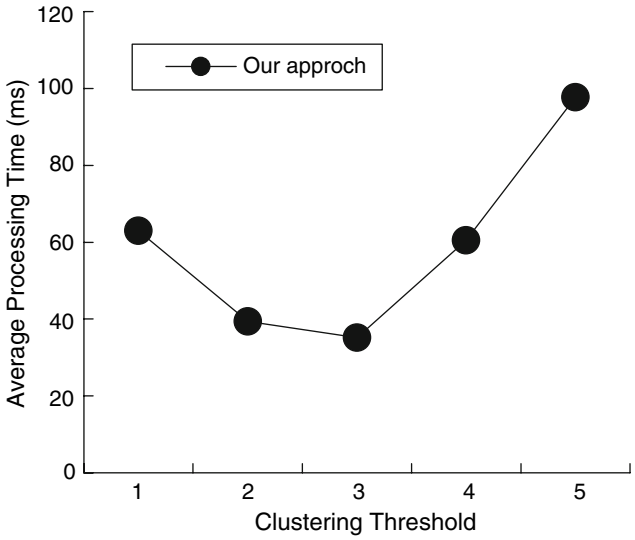

Fig. 13 The average processing time for 1,500 SQs on varied, clustering thresholds

as the number of the SQs increases, while the total processing time of our approach presents little difference among the increasing number of SQs. When the number of SQs attains 1250, the naïve approach takes 11.3 h to process the music stream while our approach only needs 84.3 min (recall that there are 168,283 events.) That is, the user has to wait for additional 10 h to receive the notification of the last song if we adopt the naïve approach. Such delay is inapplicable for the applications we introduced. On the contrary, our approach can provide reasonable response time for the users in the same condition. The results verify that the performance of our approach remains satisfactory as the number of SQs increases.

Furthermore, we tried monitoring different numbers of music streams to demonstrate the scalability of our approach in another dimension. Each music stream is formed with the same 100–songs, but in different order. The setting of the other parameters is the same as the above experiments. Figure 12 shows the total processing time for different number of SQs while monitoring one, two, and three music streams. The total processing time on three music streams is about three times of the total processing time on one music stream. This result indicates that our approach can handle multiple music streams with only a little overhead to identify the source stream of the incoming data n-grams.

6.4 Experiments on influence of parameter μ

Based on the proposed pruning mechanism, the results of query n-gram clusters may affect the performance of the pruning mechanism, and thus may also impact the performance of our approach. Choosing a suitable clustering threshold can make the individual classes tighter and maintain a reasonable number of clusters simultaneously. However, the setting of the clustering threshold is not easy because it depends on the distribution of all SQs and the streaming data, which is dynamic. Figure 13 shows the results by varying the clustering threshold in the case of 1,500 SQs. The parameters n and ε are set to 6 and 2, respectively. Our observation on this experiment is that we should choose a medium value relative to the length of the n-gram as the clustering threshold. The reason is that if μ is smaller, the number of clusters will increase, and thus the processing time of the pruning mechanism will become worse. On the other hand, if μ is larger, the clusters will become loose and thus the effect of the pruning mechanism will diminish. Moreover, the perfor-

Table 1 Time ratios on varied ε

Error bound	1	2	3	4	5
Time ratio	1	1.07	1.20	1.32	1.58

Table 2 Time ratios on varied n

Parameter n	6	8	12
Time ratio	1	2.58	4.49

mance of the merging mechanism will also be decreased if too many data n-grams pass the pruning mechanism. Note that re-clustering all the existing query n-grams whenever a new query arrives might be one solution to further reduce the number of clusters and tighten the individual clusters. However, the clustering costs so much that makes it inapplicable to the real applications.

6.5 Experiments on influence of parameter n

Since our approach is based on the concept of n-grams, choosing a suitable value of n is an important issue. To observe how this parameter influences our approach, two experiments are performed as follows. In the first experiment, we generate 1,500 SQs, set n to a constant 6, and run our approach by varying the error bound ε from 1 to 5. The error bound is set to only those values smaller than 6 because our approach will become ineffective to prune a cluster of query n-grams if ε is not below n. Taking the processing time for $\varepsilon = 1$ as the denominator, we have the relative time ratios shown in Table 1. The result shows that the time ratio grows with ε, and it means that the performance is getting worse when ε is set closer to n. The reason is that when the error bound ε is closer to length n, more data n-grams are sent to the merging mechanism and therefore increases the number of partial answers. We conclude that the length n should not be too close to the error bound ε.

From above, it seems that we should set the value of n as large as possible. However, this is not always true. In the following experiment, we generate 1,500 SQs, set ε to constant 2, and run our approach by varying the parameter n from 6, 8, to 12. Taking the processing time for $n = 1$ as the denominator, we have the relative time ratios shown in Table 2, where the time ratio grows with n. The reason is that as n grows, the number of the clusters is getting larger and the cost of distance computation becomes heavier. Therefore, we conclude that a good choice of n should be as small as possible.

From these two experiments, there is a tradeoff for the selection of the parameter n. If n is smaller, i.e., closer to ε, the overhead that the merging mechanism incurs will increase. On the other hand, if n is larger, the overhead that the pruning mechanism incurs will increase.

7 Related works

Several major research areas, DNA sequence alignment, time series analysis, repetitive pattern discovery, query-by-humming systems, content-based music retrieval, content-based audio identification, and data stream management, are related to our approach. The related works are introduced and compared with our approach as follows.

In the area of DNA sequence alignment, the researchers consider the event sequence where each event consists of a single symbol (A, T, C, or G). The *BLAST* [1,45] is the most popular tool to deal with the string matching over DNA sequences; however, false negatives may occur when using this tool. Differently, our approach guarantees no false positive and no false negative. To consider the problem of substring matching in large databases for genetic data or web data, another approach [27] is proposed to map the symbolic substrings of the data into vectors in a numerical space with the help of wavelet coefficients, and then index the coefficients for fast retrieval. If we apply it to the polyphonic music addressed in this study, the dimensions of the vectors can be even larger than the length of the original string due to the variety of events. This situation will seriously affect the performance of query processing and thus this approach is not suitable for our problem.

In the area of time series analysis, the researchers [18,39,48] consider the sequence of real numbers, in which each number represents a value at a time point. Since a time series is usually long, the computation cost can be significant. The most common solution is to map the time series into the frequency domain using Fourier transform first, and then use the first few coefficients to filter out the dissimilar data. However, these techniques cannot be directly applied to our problem because we consider a sequence of symbols instead of real numbers. Due to the differences of problem definitions and data formats, the transformation process, similarity measures (time-warping distance or Euclidean distance), and the corresponding pruning mechanisms introduced in these researches are not suitable for our problem.

The problem of finding the repetitive patterns in symbolic sequences has been discussed in [21] with the suffix tree-based solution provided. In a suffix tree, each path represents a pattern and each leaf node keeps all the positions of the corresponding pattern. After traversing the suffix tree, all the exact repetitive patterns can be extracted. In [26], two approaches are proposed to discover the non-trivial repetitive patterns, which are not contained in any other repetitive patterns. These two approaches are developed for extracting exact repetitive patterns. To find approximate repetitive patterns on monophonic music, another approach [33] is proposed to transform all the music segments in music works into histogram vectors. The histogram vector keeps the individual counts of distinct events in the music segment. The

histogram vectors are then used to estimate the lower bound of the edit distance between two music segments. The histogram vectors are first indexed in a modified R*-tree [3] and then each of them is regarded as a range query to find its approximate repetitive patterns through the index. Unfortunately, the dimensions of the histogram vectors can be huge for polyphonic music and the retrieval on the R*-tree will be inefficient. Therefore, this approach is also not suitable for the content-based retrieval over polyphonic music streams.

Providing efficient ways of content-based retrieval on a huge amount of music data in the format of SMR, such as MIDI [37] and MusicXML [41], has been an important topic for years. The main issues include the music representations and indexing methods. To represent music object, various features [46,52], such as pitch, interval, rhythm, chord, and contour, can be easily extracted from SMR. For efficient query processing, different indexing methods, such as n-gram indexing, list-based indexing, and tree-based indexing, have also been introduced [14,25].

"Query by humming" is a special case of content-based music retrieval [19,28,35]. The humming is transcribed into a sequence of discrete notes, and the "contour" information extracted from the notes is regarded as the input query. The focus of these works is how to segment a humming into discrete notes. Techniques for string matching can be adopted to enhance the performance of query processing. However, the conventional methods, such as the suffix tree [44] and the n-gram index [17] will generate a complex and costly index for searching owing to the variety of events in polyphonic music [32]. Therefore, these techniques are not suitable for our problem.

Since music works are usually polyphonic, recent researches focus on content-based retrieval for polyphonic music. In this aspect, the naïve way is to transform [53] a polyphonic work/musical segment into a monophonic one and then apply the techniques for monophonic music. Such approaches include Themefinder [30] and Meldex [36]. However, the transformation from the polyphonic music to the monophonic one is not accurate and is not applicable to every kind of music [46]. Therefore, these researches cannot solve our problem of finding the answers for the polyphonic music segments.

Without transforming the polyphonic music to the monophonic one, the system named SEMEX [31] extends the well-known shift-or algorithm [55] to find all the segments from the polyphonic music database, which contains the given monophonic query. The design of SEMEX cannot deal with the polyphonic query. By contrast, PROMS [12] directly supports the polyphonic query by using the inverted-file to record the occurrence of each note in the polyphonic music database. Unfortunately, this system is suitable for exact matching but inefficient for approximate matching. Dovey [16] proposes another algorithm for approximate matching based on dynamic programming. The definition of the approximate answer in [16] is much different from ours. The approximate answer they consider is a sequence of events of a music object that contains the query. Therefore, the users are required to give precise queries, which is not convenient to the users. Moreover, this approach suffers from the long length of a music work and the large number of music works in the database. Recently, Liu [32] proposes a novel method to efficiently retrieve k music works that contain segments most similar to the user query based on the edit distance. Since the kNN problem is quite different from the problem discussed in this paper, the pruning techniques developed in [32] cannot be applied. All the previous works focus on the content-based retrieval on static music databases. Due to the essential difference between static databases and streaming environment, all the works mentioned above cannot be applied to solve our problem. Moreover, they do not deal with the issues about simultaneous processing of multiple queries and the unbounded amount of data, either.

Different from content-based music retrieval, content-based audio identification deals with the audio files. Audio fingerprinting is the most popular technique for content-based audio identification. There are two principal components in the audio systems, i.e., computing the audio fingerprint and searching the fingerprint database. An audio fingerprint is regarded as the summary composed by the features of an audio object. Different features [6,9,23], such as Fourier coefficients, Mel-Frequency Cepstral Coefficients, and MCLT Coefficients, are adopted in the literatures. However, these coefficients are usually complicated for the users. It will be difficult for the users to set suitable thresholds for content-based audio identification if they do not understand the used fingerprint model, the discriminative information of the query, the similarity of the fingerprints in the database, and the database size [8].

To avoid linearly scanning the whole fingerprint database, certain access methods are developed in the literatures. Based on the assumption that a piece of a query's fingerprint is free from error, Haitsma et al. [22,23] propose an index by creating a lookup table for all the possible hash values of the fingerprint pieces and each entity of the lookup table points to the positions of the songs with the same fingerprint piece. For each query, the positions of all involved fingerprint pieces are located to generate the candidate songs for searching. If the number of the fingerprint pieces of a query is defined as 256, the answers retrieved will also consist of 256 successive fingerprint pieces. No insertion or deletion of a fingerprint piece is allowed. To solve our problem, the editing operators, i.e., insertion, deletion, and replacement, are necessary to measure the edit distance between the query and the data. As a result, the proposed method in [22,23] cannot solve our problem, in which the lengths of the query and the answer can be different. Moreover, the assumption is also infeasible for the applications introduced in this paper.

In [9], the audio fingerprint, called AudioDNA, is designed, which is a sequence of symbols. The matching algorithm [45] developed for biological gene sequences are adopted to speed up the comparison between the subsequences of AudioDNA from an audio stream and the fingerprints in the database. To reduce the search space, the algorithm first finds the exact matches of the short subsequences of the query and then utilizes the positions of these exact matches to generate the possible approximate answers. As a result, false negatives may occur. This algorithm is suitable for the DNA sequence comparison because a DNA sequence is a sequence of events and each event consists of a single symbol. If we apply it to the sequences in which an event can contain more than one symbol, the number of false negatives could be huge. Different from this matching algorithm, our approach guarantees no false positive and no false negative.

Various problems in the streaming environment have been discussed in the literature [20]. Data in new applications, such as web accesses, financial tickers, network packets, and sensor data [13,56], are all in the form of data streams. To support the new applications, *continuous query* (CQ) is designed to allow the users to get new results from a data stream without having to issue the same query repeatedly. Most of the data stream management systems (DSMSs) currently in use [7,11,24,40] support tuple-based data streams and CQs that have SQL-like syntax and the enhanced support for windows [10]. That is, a CQ is composed of relational operators, such as selection and join [20]. However, the querying facilities they provide cannot be used in the applications with complex data types such as the content-based retrieval over music streams. The reason is that those systems process the incoming data individually [2,10,29,34] without well addressing the issues about the temporal relationship among data, which is the characteristic of music data. Therefore, the techniques developed for the existing DSMSs cannot be applied to our problem. To keep the property of continuity of music segments, we define the music segment as a sequence of events and a query describing a music segment as the sequence query (SQ). SQ is not SQL-based but is another kind of continuous query in our approach.

Consequently, we conclude that these related works cannot deal with our problem adequately. Different from these related works, our approach is the first to solve the problem of content-based retrieval over music steams with a consideration of the two important issues in the steaming environment, i.e., simultaneous processing of multiple queries and unbounded amount of data.

8 Conclusion

Motivated by the interesting applications in the environment with music streams, we study the problem of monitoring the event stream for continuously answering a set of SQs. A novel approach extending the n-gram indexing techniques to the streaming environment is proposed, and two efficient mechanisms based on several bounding criteria are designed for pruning the candidate answers as much as possible. In the pruning mechanism, we propose a novel summarization technique for each cluster of n-grams and provide a bounding criterion to prune the clusters without any answer. In the merging mechanism, we propose a systematic way to maintain the partial answers of each query and also provide several bounding criteria to prune the partial answers as early as possible. With the reduction of the search space, both the mechanisms reduce the computation cost and space usage. The effectiveness of these mechanisms is formally proved and verified via the experiments conducted in a simulated streaming environment using real music data. Moreover, the results also show the great efficiency of our approach.

Based on the experiment results, our approach can deal with thousands of SQs on single music stream. However, as the number of SQs becomes huge, we might reach the limitation of computing resource of a personal computer. To overcome such problem, we can combine the idea of parallel processing with our approach. On one hand, the query n-gram clusters are distributed into several computers, which serve to monitor the music streams and perform the pruning mechanism. On the other hand, another computer gathers the results of approximate matches from different computers and then come out with the final results through the merging mechanism. In the near future, we will further consider the resource-limited environment. If there is no sufficient resource to support the parallel processing under the real-time requirement, a load shedding mechanism that can selectively ignore the incoming data n-grams or focus on the query n-grams of more importance will be required.

If there are many music streams to be monitored, the number of data n-grams to be processed in a unit time could be large. For efficiency, we can process a batch of the data n-grams at one time. For a number of data n-grams that arrive during a time period, we can also divide them into clusters as we process the query n-grams. In the pruning mechanism, we can estimate the minimum distance between the summarization of the data n-gram cluster and the summarization of the query n-gram cluster to see whether we can prune the entire cluster. The more similar the data n-grams are, the less computation will be needed. However, the clustering of the newly added data n-grams suffers from run-time overhead. The investigation of the tradeoff between the computation sharing and the overhead of clustering is one of our future works.

References

1. Altschul, S., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. (1990)

2. Avnur, R., Hellerstein, J.M.: Eddies: continuous adaptive query processing. In: Proceedings of ACM SIGMOD International Conference on Management Data (2000)
3. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In: Proceedings of ACM SIGMOD International Conference on Management of Data (1990)
4. Bellini, P., Barthelemy, J., Nesi, P., Zoia, G.: A proposal for the integration of symbolic music notation into multimedia frameworks. In: Proceedings of International Conference on the Web Delivering of Music (2004)
5. Bellini, P., Nesi, P., Zoia, G.: Symbolic music representation in MPEG. IEEE Multimedia (2005)
6. Burges, C.J.C., Platt, J.C., Jana, S.: Distortion discriminant analysis for audio fingerprinting. IEEE Trans. on Speech Audio Process. (2003)
7. Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.: Monitoring streams — a new class of data management applications. In: Proceedings of International Conference on Very Large Data Bases (2002)
8. Cano, P., Batlle, E., Kalker, T., Haitsma, J.: A review of algorithms for audio fingerprinting. In: International Workshop on Multimedia Signal Processing (2002)
9. Cano, P., Batlle, E., Mayer, H., Neuschmied, H.: Robust sound modeling for song detection in broadcast audio. In: Processing of International Conference on AES (2002)
10. Chandrasekaran, S., Franklin, M.J.: Streaming Queries over Streaming Data. In: Proceedings of International Conference on Very Large Data Bases (2002)
11. Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: NiagraCQ: a scalable continuous query system for internet databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data (2000)
12. Clausen, M., Engelbrecht, R., Meyer, D., Schmitz, J.: Proms: a web-based tool for searching in polyphonic music. In: Proceedings of International Symposium on Music Information Retrieval (2000)
13. Cranor, C., Gao, Y., Johnson, T., Shkapenyuk, V., Spatscheck, O.: Gigascope: high performance network monitoring with an SQL interface. In: Proceedings of ACM SIGMOD Conference on Management of Data (2003)
14. Crochemore, M., Rytter, W.: Text Algorithms. Oxford University Press, Oxford (1994)
15. Dannenberg, R.B., Hu, N.: Pattern discovery techniques for music audio. In: Proceedings of International Symposium on Music Information Retrieval (2002)
16. Dovey, M.J.: A technique for "regular expression" style searching in polyphonic music. In: Proceedings of International Symposium on Music Information Retrieval (2001)
17. Downie, S., Nelson, M.: Evaluation of a simple and effective music information retrieval method. In: Proceedings of ACM International Conference on Research and Development in Information Retrieval (2000)
18. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data (1994)
19. Ghias, A., Logan, J., Chamberlin, D., Smith, B.C.: Query by humming: music information retrieval in an audio database. In: Proceedings of ACM Multimedia (1995)
20. Golab, L., Özsu, M.T.: Issues in data stream management. ACM SIGMOD Record (2003)
21. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, Cambridge (1997)
22. Haitsma, J., Kaller, T., Oostveen, J.: Robust audio hashing for content identification. In: Proceedings of IEEE International Workshop on Content-based Multimedia Indexing (2001)
23. Haitsma, J., Kalker, T.: A highly robust audio fingerprinting system. In: Proceedings of International Symposium on Music Information Retrieval (2002)
24. Hellerstein, J., Franklin, M., Chandrasekaran, S., Deshpande, A., Hildrum, K., Madden, S., Raman, V., Shah, M.A.: Adaptive query processing: technology in evolution. IEEE Data Eng. Bull. (2000)
25. Hsu, J.L., Chen, A.L.P.: Building a platform for performance study of various music information retrieval approaches. In: Proceedings of International Symposium on Music Information Retrieval (2001)
26. Hsu, J.L., Liu, C.C., Chen, A.L.P.: Discovering non-trivial repeating patterns in music data. IEEE Trans. Multimedia (2001)
27. Kahveci, T., Singh, A.K.: An efficient index structure for string databases. In: Proceeding of International Conference on Very Large Data Bases (2001)
28. Kageyama, T., Mochizuki, K., Takashima, Y.: Melody retrieval with humming. In: Proceedings of International Computer Music Conference (1993)
29. Kang, J., Naughton, J.F., Viglas, S.D.: Evaluating window joins over unbounded streams. In: Proceedings of IEEE International Conference on Data Engineering (2003)
30. Kornstadt, A.: Themefinder: A Web-Based Melodic Search Tool. In: Computing in Musicology, vol. 11. MIT Press, Cambridge (1998)
31. Lemstrõm, K., Perttu, S.: SEMEX – An efficient music retrieval prototype. In: Proceedings of International Symposium on Music Information Retrieval (2000)
32. Liu, N.H., Wu, Y.H., Chen, A.L.P.: Efficient kNN Search in Polyphonic Music Databases Using a Lower Bounding Mechanism. ACM Multimedia Syst. J. (2005)
33. Liu, N.H., Wu, Y.H., Chen, A.L.P.: An efficient approach to extracting approximate repeating patterns in music databases. In: Proceedings of International Conference on Database Systems for Advanced Applications (2005)
34. Madden, S., Shah, M., Hellerstein, J.M., Raman, V.: Continuous adaptive continuous queries over streams. In: Proceedings of ACM SIGMOD International Conference on Management Data (2002)
35. McNab, R.J., Smith, L.A., Witten, I.H., Henderson, C.L., Cunningham, S.J.: Towards the digital music library: tune retrieval from acoustic input. In: Proceedings of Digital Libraries Conference (1996)
36. McNab, R.J., Smith, L.A., Bainbridge, D., Witten, I.H.: The new zealand digital library MELodyinDEX. Digi. Libr. Mag. (1997)
37. MIDI Manufacturers Association, Los Angeles, California: The Complete Detailed MIDI 1.0 Specification (1996)
38. Mongeau, M., Sankoff, D.: Comparison of Musical Sequences. In: Computer and the Humanities (1990)
39. Moon, Y.S., Whang, K.Y., Han, W.S.: General match: a subsequence matching method in time-series databases based on generalized windows. In: Proceedings of ACM SIGMOD International Conference on Management of Data (2002)
40. Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., Varma, R.: Query processing, approximation, and resource management in a data stream management system. In: Proceedings of Biennial Conference on Innovative Data Systems Research (2003)
41. MusicXML, http://www.recordare.com/
42. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. (2001)
43. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. (1970)

44. Park, S., Chu, W.W., Yoon, J., Hsu, C.: Efficient searches for similar subsequences of different lengths in sequence databases. In: Proceedings of International Conference on Data Engineering (2000)
45. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence comparison. In: Proc. Nat. Acad. Sci. (1998)
46. Pickens, J.: A survey of feature selection techniques for music information retrieval. In: Proceedings of International Symposium on Music Information Retrieval (2001)
47. Pirkola, A., Keskustalo, H., Leppanen, E., Kansala, A.P., Jarvelin, K.: Targeted s-gram matching: a novel n-gram matching technique for cross- and monolingual word form variants. Inf. Res. (2002)
48. Rafiei, D., Mendelzon, A.: Similarity-based queries for time series data. In: Proceedings of ACM SIGMOD International Conference on Management of Data (1997)
49. Robertson, A.M., Willett, P.: Applications of n-grams in textual information systems. J. Doc. (1998)
50. Root Cause Analysis Handbook: A Guide to Effective Incident Investigation, by Risk and Reliability Division, ABS Group, Inc. Root Cause Map, ABSG Consulting Inc. (1999)
51. Salton, G.: Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer. Addison Wesley, Reading (1989)
52. Selfridge-Field, E.: Conceptual and representational issues in melodic comparison. Melodic similarity: concepts, procedures, and applications (Computing in Musiccology:11), The MIT Press, Cambridge (1998)
53. Uitdenbogerd, A., Zobel, J.: Melodic Matching Techniques for Large Music Databases. In: Proceedings of ACM Multimedia (1999)
54. Ukkonen, E.: Algorithms for approximate string matching. Inf. Control (1985)
55. Wu, S., Manber, U.: Fast text searching allowing errors. Commun. ACM (1992)
56. Yao, Y., Gehrke, J.: Query Processing for Sensor Networks. In: Proceedings of Conference on Innovative Data Systems Research (2003)
57. Zobel, J., Dart, P.: Finding approximate matches in large lexicons. In: Software—practice and experiences (1995)