DETECTING CHANGES IN USER-CENTERED MUSIC QUERY STREAMS

Hua-Fu Li[*a], *Man-Kwan Shan*[b], *and Suh-Yin Lee*[a]

[a] Department of Computer Science, National Chiao-Tung University, Hsinchu 300, Taiwan
[b] Department of Computer Science, National Chengchi University, Taipei 116, Taiwan
{hfli, sylee}@csie.nctu.edu.tw; mkshan@cs.nccu.edu.tw

ABSTRACT

In this paper, we propose an efficient algorithm, called MQS-change (changes of Music Query Streams), to detect the changes of maximal melody structures in user-centered music query streams. Two music melody structures (set of chord-sets and string of chord-sets) are maintained and four melody structure changes (positive burst, negative burst, increasing change and decreasing change) are monitored in a new data structure MSC-list (a list of Music Structure Changes). Experiments show that MQS-change algorithm is an online, single-pass approach to detect the changes of music melody structures over continuous music query streams.

1. INTRODUCTION

Recently, database and data mining communities have been focused on a new data model, where data arrives in the form of *continuous streams*. It is often refer to *data streams* or *streaming data*. Many applications generate large amount of data streams in real time, such as sensor data generated from sensor networks, transaction flows in retail chains, Web click and record streams in Web applications, performance measurement in network monitoring and traffic management, call records in telecommunications, etc. Mining streaming data differs from mining traditional static data sets in two main aspects [2] [6]:

- The volume of a continuous stream over its lifetime could be huge and fast changing.
- The queries require timely answers, and the response time is short.

Hence, it is not possible to store all the data in main memory or even in secondary storage. This motivates the design for in-memory *summary* data structure with small memory footprints that can support both one-time and continuous queries. In other words, online algorithms of mining data streams have to sacrifice the correctness of their analysis results by allowing some counting errors, i.e., *approximate* results, and only have *one pass* over the data.

Mining music data is one of the most important research issues in data mining. Although several techniques have been developed recently for discovering and analyzing the content of *static* music data [3][7][12][13], new techniques are needed to analyze and discover the content of *streaming* music data. Li et al. [9][10] proposed efficient single-pass algorithms to find maximal frequent melody structures and closed frequent melody structures over continuous music query streams. The problem comes from the context of online music-downloading services (such as *iTunes*, *Kuro* [14] and *KKBOX* [15]), where the streams in question are streams of queries, i.e., music-downloading requests, sent to the server, and we are interested in finding the useful music melody structures requested by most customers during some period of time. With the computation model of music melody streams presented in Fig. 1, the melody stream processor and the summary data structure are two major components in such a streaming environment. The user query processor receives user queries in the form of <Timestamp, Customer-ID, Music-ID>, and then transforms the queries into music data (i.e., melody sequences) in the form of <Timestamp, Customer ID, Music-ID, Melody-Sequence> by querying the music database. Note that the buffer can be optionally set for temporary storage of recent music melodies from the music melody streams.

With data streams, people are more often interested in mining queries such like "*Compared to the history, what are the distinct features of the current status?*", " *What are the most popular melody structures in the last four hours*?" and "*What are the relatively stable factors over time?*" To answer the above queries, we have to examine the changes of streaming data [5][11]. Hence, this paper studies a new problem of how to detect the changes of maximal melody structures over user-centered music query streams. An efficient single-pass algorithm MQS-change is proposed to detect the changes of music query streams. Experiments show that the proposed algorithm is an effective algorithm to detect the changes in data streams efficiently.

The remainder of the paper is organized as follows. The problem is defined in Section 2. In Section 3, we describe the design of the MQS-change algorithm for detecting changes of music query streams. Experiments are discussed in Section 4. Finally, we conclude our work in Section 5.

[*] *Corresponding author*. Fax: 886-3-5721490. E-mail: hfli@csie.nctu.edu.tw

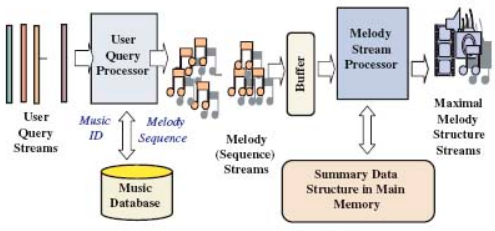

Fig. 1. Computation model for music melody streams.

2. PROBLEM DEFINITION

In this section, we describe several features of music data used in this paper and define the problem of detecting changes of user-centered music query streams. For the basic terminologies on music, we refer to [8][12].

Definition 1 A *chord* is the sounding combination of three or more notes at the same time. A *note* is a single symbol on a musical score, indicating the pitch and duration of what is to be sung and played. A *chord-set* is a set of chords.

Definition 2 The *type I melody structure* is represented as a set of chord-sets.

Definition 3 The *type II melody structure* is represented as a string of chord-sets.

Definition 4 Let $\Psi = \{i_1, i_2, \ldots, i_n\}$ be a set of *chord-sets*, called *items* for simplicity. A *melody sequence S* with *m* chord-sets is denoted by $S_m = \langle x_1 x_2 \ldots x_m \rangle$, where $x_i \in \Psi$, $\forall i = 1,2, \ldots, m$. A *basic window w* is consecutive subsequence of *t* melody sequences. The *size* of a basic window $|w|$ is *t*.

Definition 5 A music melody stream (MMS) is an infinite sequence of basic windows, where each window w_i is associated with a window identifier *i*, and *z* is the identifier of the ''*latest*'' window w_z, i.e., MMS = $[w_1, w_2, \ldots, w_z)$. The *current length* of MMS, written as |MMS|, is *zt*, i.e., $|\text{MMS}| = |w_1| + |w_2| + \cdots + |w_z|$. These windows arrive in some order (implicitly by arrival time or explicitly by timestamp), and may be seen only once.

Definition 6 A set $Y \subseteq \Psi$ is called an *itemset*, i.e., *a set of chord-sets*. A *k-itemset* is represented by $(y_1 y_2 \cdots y_k)$. The *support* of an itemset *Y*, denoted as **sup**(*Y*), is the number of melody sequences containing *Y* as a *subset* in the MMS so far. An itemset *Y* is *frequent* if $\textbf{sup}(Y) \geq s \cdot |\text{MMS}|$, where *s* is a user-defined minimum support threshold in the range of [0, 1].

Definition 7 A string *Z* is called an *item-string*, i.e., *a string of chord-sets*. A *k-item-string* is represented by $\langle z_1 z_2 \ \ldots \ z_k \rangle$, where $z_i \in \Psi$, $\forall i = 1, 2, \ldots, k$. The *support* of an item-string *Z*, denoted as **sup**(*Z*), is the number of melody sequences containing *Z* as a *substring* in the MMS so far. An item-string is *frequent* if $\textbf{sup}(Z) \geq s \cdot |\text{MMS}|$.

Definition 8 A frequent itemset is called a *maximal frequent itemset* (**MFI**) if it is not a subset of any other frequent itemsets.

Definition 9 A frequent item-string is called a *maximal frequent item-string* (**MFS**) if it is not a substring of any other item-strings.

Definition 10 A maximal frequent itemset *P* is called *positive itemset burst* (**PIB**) if its $\textbf{sup}(P)^z - \textbf{sup}(P)^{z-1} \geq \partial_{\text{MFI}}$, where ∂_{MFI} is a user-specified itemset burst threshold in the range of [0, 1], $\textbf{sup}(P)^z$ is the estimated support of *P* from window w_1 to window w_z.

Definition 11 A maximal frequent itemset *P* is called *negative itemset burst* (**NIB**) if its $\textbf{sup}(P)^{z-1} - \textbf{sup}(P)^z \geq \partial_{\text{MFI}}$.

Definition 12 A maximal frequent item-string *Q* is called *positive item-string burst* (**PSB**) if its $\textbf{sup}(Q)^z - \textbf{sup}(Q)^{z-1} \geq \partial_{\text{MFS}}$, where ∂_{MFS} is a user-specified item-string burst threshold in the range of [0, 1], $\textbf{sup}(Q)^z$ is the estimated support of *Q* from window w_1 to window w_z.

Definition 13 A maximal frequent item-string *Q* is called *negative item-string burst* (**NSB**) if its $\textbf{sup}(Q)^{z-1} - \textbf{sup}(Q)^z \geq \partial_{\text{MFS}}$.

Definition 14 A maximal frequent itemset *P* is called *increasing changed itemset* (**ICI**) if $\partial_{\text{MFI}} > (\textbf{sup}(P)_{i+1} - \textbf{sup}(\text{P})_i) \geq \varepsilon_{\text{MFI}}$, $\forall i$, $i = z\text{-}h_1\text{+}1, z\text{-}h_1\text{+}2, \ldots, z$, where ε_{MFI} is a user-specified increasing changed itemset threshold in the range of [0, 1], and h_1 is a number of basic windows defined by user.

Definition 15 A maximal frequent item-string *Q* is called *increasing changed item-string* (**ICS**) if $\partial_{\text{MFS}} > (\textbf{sup}(Q)_{j+1} - \textbf{sup}(Q)_j) \geq \varepsilon_{\text{MFS}}$, $\forall j$, $j = z\text{-}h_2\text{+}1, z\text{-}h_2\text{+}2, \ldots, z$, where ε_{MFS} is a user-specified increasing changed item-string threshold in the range of [0, 1], and h_2 is a number of basic windows defined by user.

Definition 16 A maximal frequent itemset *P* is called *decreasing changed itemset* (**DCI**) if $\partial_{\text{MFI}} > (\textbf{sup}(Q)_j - \textbf{sup}(Q)_{j+1}) \geq \gamma_{\text{MFI}}$, $\forall j$, $j = z\text{-}h_1\text{+}1, z\text{-}h_1\text{+}2, \ldots, z$, where γ_{MFI} is a user-specified decreasing changed item-string threshold in the range of [0, 1], and h_1 is a number of basic windows defined by user.

Definition 17 A maximal frequent item-string *Q* is called *decreasing changed item-string* (**DCS**) if $\partial_{\text{MFS}} > (\textbf{sup}(Q)_j - \textbf{sup}(Q)_{j+1}) \geq \gamma_{\text{MFS}}$, $\forall j$, $j = z\text{-}h_2\text{+}1, z\text{-}h_2\text{+}2, \ldots, z$, where γ_{MFS} is a user-specified decreasing changed item-string threshold in the range of [0, 1], and h_2 is a number of basic windows defined by user.

Problem Definition Given a MMS, *s*, ∂_{MFI}, ∂_{MFS}, ε_{MFI}, ε_{MFS}, γ_{MFI}, and γ_{MFS}, the problem of detecting changes in user-centered music query streams is to maintain the set of MFI and MFS, and to detect the set of PIB, NIB, PSB, NSB, ICI, ICS, DCI, and DCS, by one scan of a continuous user-centered music query stream.

3. DETECTING CHANGES IN USER-CENTERED MUSIC QUERY STREAMS

In this section, a new summary data structure MSC-list (a list of Music Structure Changes) is developed to maintain the essential information about MFI, MFS, PIB, NIB, PSB, NSB, ICI, ICS, DCI, and DCS with their supports embedded in the individual window of the current MMS. An online, single-pass algorithm MQS-change (changes of Music Query Streams) is proposed to mine the changes from user-centered music query streams.

3.1. A New Summary Data Structure MSC-list

MSC-list consists of two temporal lists, MFI-list and MFS-list, where MFI-list is a list of entries which contains current maximal frequent itemsets, and MFS-list is a list of entries which maintains maximal frequent item-strings so far. Each entry of MFI-list consists of two fields: *pattern-id Y* and *support-list Y.support-list*, where pattern-id is a unique identifier of this maximal frequent itemset, and support-list is composed of a list of (**sup**(*Y*), *i*), where *i* is the window identifier of window w_i containing the itemset *e*. For example, an entry <*abcd*, (30%, 1), (37%, 2), (46%, 3), (70%, 4)> of MFI-list indicates that the itemset *abcd* is a maximal frequent itemset and its estimated support is 30% in window w_1, 37% in w_2, 46% in w_3, and 70% in w_4. Assume that the ∂_{MFI} is 0.2 (i.e., 20%), ε_{MFI} = 0.05 (i.e., 5%), and h_1 be 3 (i.e., 3 consecutive windows). Hence, the pattern *abcd* is an *increasing changed itemset* from windows w_1 to w_3, and has a *positive itemset burst* in window w_4. Eeach entry of MFS-list also consists of two fields: *pattern-id Z* and *support-list Z.support-list*, where pattern-id is a unique identifier of this maximal frequent item-string, and support-list is composed of a list of (**sup**(*Z*), *i*), where *i* is the identifier of window w_i containing the item-string *Z*. In the following, we use the term *maximal frequent pattern* (**MFP**) to substitute the maximal frequent itemset and maximal frequent item-string.

Two operations are used to maintain the MSC-list:

(1) Update MSC-list: For each entry <*pattern-id*, *support-list*> of MSC-list, MQC-change algorithm updates the support-list of this entry, i.e., append a new support record to the support-list. If there are changes happen (according to the definition 11 through definition 17), the pattern is inserted into a *temporal change output queue* (*TCOQ*). If an entry *e* is not a MFP, i.e., **sup**(*e*) < s·|MMS|, the entry is deleted from the current MSC-list.

(2) New MSC-list: if MQC-change find a maximal frequent pattern *P* from the current window w_z and $P \notin$ MSC-list, and **sup**(*P*) ≥ s·t, where *s* is the minimum support threshold, and *t* is the window size, a new entry of the form <*P*, (**sup**(*P*), *z*)>, where *z* is the current window identifier, is created in the current MSC-list.

3.2. The Proposed Algorithm

The proposed MQS-change algorithm is composed of four steps. First, MQS-change repeatedly reads a window of melody sequences into available main memory. Second, the maximal frequent itemsets and maximal frequent item-strings in the current window are mined using MMS_{LMS} algorithm [10], and added into MSC-list with their potential supports computed. Third, the set of MFIs and MFSs are maintained in the current MSC-list, and the changes are verified by MQS-change. Finally, MQS-change will return the changed patterns immediately if the user-centered music query stream has a change.

3.2.1. *Description of MMS_{LMS} Algorithm*

We use an illustrative example to describe the main idea of MMS_{LMS} algorithm. Let a window w_j of MMS be <*acdef*>, <*abe*>, <*cef*>, <*acdf*>, <*cef*>, and <*df*>, and minimum support threshold *s* be 0.5, where *a*, *b*, *c*, *d*, *e*, and *f* are chord-sets.

First, MMS_{LMS} algorithm projects each melody sequence into a set of *item-suffix* melody sequences. For example, the first melody sequence <*acdef*> is projected into five item-suffix melody sequences, i.e., <*f*>, <*ef*>, <*def*>, <*cdef*>, and <*acdef*>.

Second, these item-suffix melody sequences are inserted into a prefix tree-based summary data structure. The result of processing first melody sequence <*acdef*> is shown in Fig. 2.

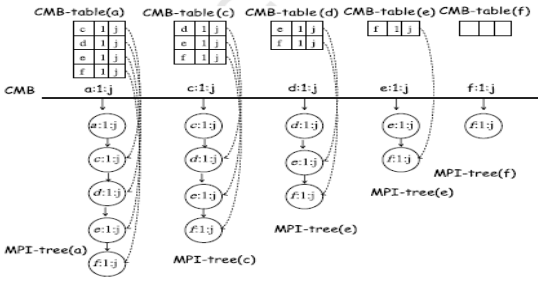

Fig. 2. Processing first melody sequence <*acdef*> by MMS_{LMS}

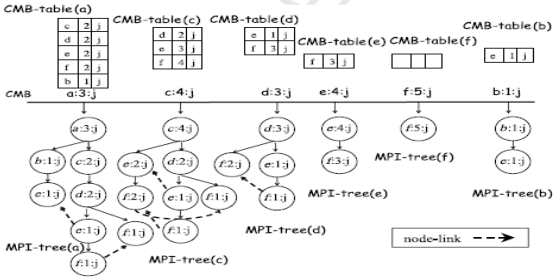

Fig. 3. Processing the window w_j by MMS_{LMS}

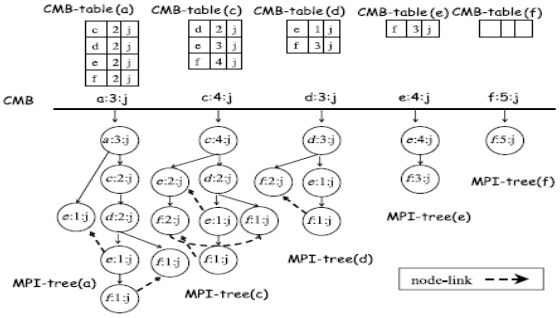

Fig. 4. Pruning infrequent information by MMS_{LMS}

After processing the window w_j (the result is shown in Fig. 3), MMS_{LMS} prunes infrequent patterns, i.e., item *b* and its super-patterns, by traversing the summary data structure proposed by MMS_{LMS} algorithm. The result after pruning infrequent patterns is shown in Fig. 4.

Finally, MMS_{LMS} algorithm uses top-down maximal pattern discovery technique to find the set of MFIs, (*a*), (*cef*) and (*df*), and MFSs, (*a*), (*c*), (*d*), and (*ef*). More detail about the MMS_{LMS} algorithm can be found in [10].

3.2.2. *Algorithm MQS-change*

The MQS-change algorithm is shown in Fig. 5.

Algorithm MQS-change

Input: (1) *MMS*, (2) *s*, (3) ∂_{MFI}, (4) ∂_{MFS}, (5) ε_{MFI}, (6) ε_{MFS}, (7) γ_{MFI}, (8) γ_{MFS}, (9) h_1, (10) h_2.
Output: Changes (PIB, NIB, PSB, NSB, ICI, ICS, DCI, and DCS).

```
begin
  MSC-list = NULL;
  Repeat:
    for each window w_i in MMS do  // ∀i =1,2, …, z.
      Mine MFPs from w_i by using MMS_LMS algorithm;
      for each MFP of w_i do
        if MFP ∈ MSC-list then
          Update MSC-list;
          Output TCOQ;
        else
              New MSC-list;
        end if
      end for
      for each entry e in the MSC-list do  // Pruning
        for each sup(e) in e.support-list do
          if sup(e)^i < s·(z–i+1) then // i is the window identifier
            Delete the entry (sup(e), i) from e.support-list;
          end if
        end for
        if sup(e) < s·|MMS| then  // e is not a MFP
          Delete e from MSC-list;
        end if
      end for
    end for
end
```

Fig. 5. MQS-change algorithm

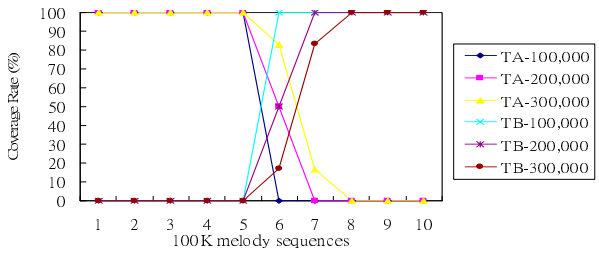

Fig 6. Coverage rate for *T5.I4.D1000K-AB*

4. PERFORMANCE EVALUATION

In this section, the performance of MQS-change algorithm is analyzed by a synthetic music query stream *T5.I4.D1000K-AB*, where three parameters denote the average melody sequence size (*T*), the average maximal frequent pattern size (*I*), and the total music melody sequences (*D*), respectively. The data is generated by the IBM synthetic data generator proposed by Agrawal and Srikant [1]. *T5.I4.D1000K-AB* consists of two consecutive subparts *TA* and *TB*. *TA* denotes a set of melody sequences generated by a set of chord-sets *A* while *TB* denotes a set of sequences generated by a set of chord-sets *B*. There is no common chord-sets between *TA* and *TB*. *TA*-100,000 indicates that the size of the tested window in *TA* is 100,000 melody sequences. Due to the space limitation, we only discuss the adaptability of the proposed algorithm in this section.

We use the coverage rate [4] to evaluate the adaptability of the MQS-change algorithm. The result is shown in Fig. 6. As the size of a window becomes smaller, the MQS-change adapts more rapidly the change of recent information between the two different subparts of *T5.I4.D1000K-AB*.

5. CONCLUSIONS

In this paper, we propose a new online algorithm MQS-change (changes of Music Query Streams) to maintain two music melody structures (sets of chord-sets and string of chord-sets) and to detect three music melody structure changes (significant pattern bursts, increasing changed patterns and decreasing changed patterns) from a continuous user-centered music query stream. A new summary data structure MSC-list (a list of Music Structure Changes) is developed to maintain the essential information about the maximal melody structures of music query streams so far. Based on our knowledge, MQS-change algorithm is the first online, single-pass method to detect the changes in a continuous user-centered music query stream.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in: Proc. VLDB, pp. 487–499, 1994.
[2] B. Babcock, S. Babu, M. Data, R. Motwani and J. Widom, "Models and issues in data stream systems," in: Proc. PODS, pp. 1–16, 2002.
[3] V. Bakhmutora, V. U. Gusev and T.N. Titkova, "The search for adaptations in song melodies," Computer Music Journal, 21 (1), 58–67, 1997.
[4] J. H. Chang and W. S. Lee, "Finding recent frequent itemsets adaptively over online data streams," in: Proc. SIGKDD, pp. 487-492, 2003.
[5] G. Dong, J. Han, L.V.S. Lakshmanan, J. Pei, H. Wang and P.S. Yu, "Online mining of changes from data streams: research problems and preliminary results," in: Proc. ACM SIGMOD-MPDS, 2003.
[6] M. M. Gaber, A. Zaslavsky and S. Krishnaswamy, "Mining data streams: a review," ACM SIGMOD Record, 34(1), June 2005.
[7] J.-L. Hsu, C.-C. Liu and A.L.P. Chen, "Discovering nontrivial repeating patterns in music data," IEEE Transactions on Multimedia, 3 (3), 311–325, 2001.
[8] G.T. Jones, Music Theory. Harper & Row, Publishers, New York., 1974.
[9] H.-F. Li, S.-Y. Lee and M.-K. Shan, "Mining frequent closed structures in streaming melody sequences," in: Proc. ICME, 2004.
[10] H.-F. Li, S.-Y. Lee and M.-K. Shan, "Online mining maximal frequent structures in continuous landmark melody streams," Pattern Recognition Letters, 26 (11), 1658-1674, August 2005.
[11] H.-F. Li, S.-Y. Lee and M.-K. Shan, "Online mining changes of items over continuous append-only and dynamic data streams," Journal of Universal Computer Sciences, 11(8), 1411-1425, 2005.
[12] M.-K. Shan and F.-F. Kuo, "Music style mining and classification by melody," IEICE Transactions on Information and Systems, E86-D (4), 655–659, 2003.
[13] A. Yoshitaka and T. Ichikawa, "A survey on content-based retrieval for multimedia databases," IEEE Transactions on Knowledge and Data Engineering, 11 (1), 81–93, 1999.
[14] www.music.com.tw
[15] www.kkbox.com.tw