ONLINE MINING OF RECENT MUSIC QUERY STREAMS

Hua-Fu Li[*a], *Chin-Chuan Ho*[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, hocc}@csie.nctu.edu.tw; mkshan@cs.nccu.edu.tw

ABSTRACT

Mining multimedia data is one of the most important issues in data mining. In this paper, we propose an online one-pass algorithm to mine the set of frequent temporal patterns in online music query streams with a sliding window. An effective bit-sequence representation is used to reduce the processing time and memory needed to slide the windows. Experiments show that the proposed algorithm only needs a half of memory requirement of original music query data, and just scans the data once.

1. INTRODUCTION

In recent years, several emerging applications warrant mining and discovering frequent patterns in data streams, e.g., sensor data generated from sensor networks, online transaction flows in retail chains, streaming Web click-sequences and records in Web applications, performance measurement in network monitoring and traffic management, and inventory stock monitoring. Data streams are characterized by two features [2] [4]:

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

These features raise new challenges for mining data streams, such as it is not possible to store all the streaming 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. Furthermore, an online method for mining data streams has to sacrifice the correctness of its analysis results by allowing some counting errors, i.e., it generates *approximate* results, and only has *one pass* over the data.

Mining music data is one of the most important research issues of multimedia data mining. Although several techniques have been developed recently for discovering and analyzing the content and usage of *static music data* [3][5][6][9][10], new techniques are needed to analyze and discover the content and usage of *streaming music data*. The problem of mining streaming music data comes from the context of online music-downloading services (such as *iTunes*, *Kuro* [11] and *KKBOX* [12]), 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 processing model of music query streams presented in Figure 1, the melody stream processor and the summary data structure are two major components. 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 component *Buffer* can be optionally set for temporary storage of recent music melody sequences from the music query streams.

Recently, Li et al. [7][8] proposed the novel online algorithms to find the complete set of maximal frequent melody structures and closed frequent melody structures over the *entire history* of a continuous music query stream. Generally, knowledge embedded in a data stream is more likely to be changed as time goes by. Mining the recent interesting patterns of an online data stream can provide valuable information for the analysis of the data stream. Hence, in this paper, we propose a new online algorithm called FTP-MQS (Frequent Temporal Patterns of Music Query Streams) to mine the set of frequent temporal patterns, i.e., frequent melody structures, over a continuous music query stream with a *sliding window*. Experiments show that the proposed algorithm is an efficient online method for mining music query streams.

The rest of the paper is organized as follows. The problem is defined in Section 2. In Section 3, we introduce the design of the proposed algorithm. Experimental results are discussed in Section 4. We conclude the paper in Section 5.

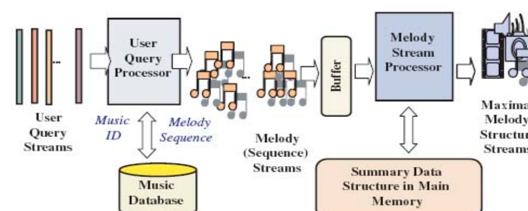

Figure 1: Computation model for music query streams

2. PROBLEM DEFINITION

In this section, several features of music data are described and the problem is defined. The basic terminologies on music used in this paper are referred to [6][9]. 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.

Let $\Psi = \{i_1, i_2, \ldots, i_n\}$ be a set of *chord-sets*, called **items** for simplicity. An **itemset** is a set of items, i.e., a set of chord-sets. A

[*] *Corresponding author*. Email:hfli@csie.nctu.edu.tw

***k*-itemset** is an itemset with k items, denoted as $(x_1, x_2, \ldots, x_k)$. For brevity, the commas are omitted. For example, a 3-itemset (a, b, c) is written as (abc). A **melody sequence stream**, *MSS*, is a sequence of incoming melody sequences, $[m_1, m_2, \ldots, m_N)$, where a melody sequence m_i is composed of a melody sequence identifier (MSID) m_i and an itemset, and N is an unknown largest number of melody sequences that will arrive. The sequence of w recent melody sequences of *MSS* is called the **sliding window** (SW) of *MSS*, i.e., SW = $[m_{N-w+1}, m_{N-w+2}, \ldots, m_N]$. The **support** of an itemset X, denoted as $\mathbf{sup}(X)$, is the number of melody sequences in SW containing X as a subset. An itemset X is a **frequent temporal pattern** (FTP), if and only if $\mathbf{sup}(X) \geq s \cdot w$, where s is a user-defined minimum support threshold in the range of [0, 1]. An itemset X is called **infrequent temporal pattern** (ITP), if and only if $\mathbf{sup}(X) < s \cdot w$.

Problem Definition Given a melody sequence stream *MSS*, a user-defined minimum support threshold s, and the size of sliding window w, the problem of online mining of user-centered music query streams is to discover the set of frequent temporal patterns by one scan of the w recent melody sequences of MSS.

Example 1 Let the first four melody sequences in a melody sequence stream be <m_1, (acd)>, <m_2, (bce)>, <m_3, $(abce)$>, and <m_4, (be)>, where m_1, m_2, m_3, and m_4 are melody sequence identifiers and a, b, c, d, and e are chord-sets. Let the size of sliding window w be 3 and the user-defined minimum support threshold s be 0.6. Hence, the current melody sequence stream consists of *two* sliding windows, i.e., $SW_1 = [m_1, m_2, m_3]$ and $SW_2 = [m_2, m_3, m_4]$, where first window SW_1 contains the melody sequences m_1, m_2, and m_3, and the second window SW_2 contains the sequences m_2, m_3, and m_4. The example is shown in Figure 2.

Melody Sequence Stream	FTPs in SW_1	FTPs in SW_2
<m_1, (acd)> <m_2, (bce)> <m_3, $(abce)$> <m_4, (be)>	(a), (b), (c), (e), (ac), (bc), (be), (ce), (bce)	(b), (c), (bc), (be), (ce), (bce)

A melody sequence stream is formed by melody sequences arriving in series

Figure 2: An example melody sequence stream and the frequent temporal patterns in two consecutive sliding windows

In Figure 2, the frequent temporal patterns in SW_1 are (a), (b), (c), (e), (ac), (bc), (be), (ce), and (bce), and the frequent ones in SW_2 are (b), (c), (d), (bc), (be), (ce), and (bce). In this example, we can find that $\{(a), (e), (ac)\}$ are frequent temporal patterns in SW_1, but are not frequent ones in SW_2.

3. ONLINE MINING OF RECENT MUSIC QUERY STREAMS

3.1. Bit-Sequence Representation of 1-Itemsets

In the proposed algorithm, for each item X in the current sliding window, a *bit-sequence* with w bit, denoted as $\mathbf{Bit}(X)$, is constructed. If an item X is in the i-th music sequence of current sliding window, the i-th bit of $\mathbf{Bit}(X)$ is set to be 1; otherwise, it is set to be 0. The process is called *bit-sequence transform*.

Example 2 Consider the melody sequence stream in Figure 2 and assume that the size of sliding window is 3. The sliding window SW_1 consists of three consecutive melody sequences: <m_1, (acd)>, <m_2, (bce)>, and <m_3, $(abce)$>, and five items (chord-sets): a, b, c, d and e. Because item a appears in the 1st and 3rd melody sequences of SW_1, the bit-sequence of a, $\mathbf{Bit}(a)$, is 101. Similarly, $\mathbf{Bit}(b)$ = 011, $\mathbf{Bit}(c)$ = 111, $\mathbf{Bit}(d)$ = 100, and $\mathbf{Bit}(e)$ = 011.

3.2. The Proposed Algorithm FTP-MQS

In this section, based on the representation of appearing items, an efficient algorithm FTP-MQS is introduced. FTP-MQS consists of three phases: *window initialization* phase, *window sliding* phase, and *frequent temporal patterns generation* phase.

3.2.1. Window Initialization Phase

The phase is activated while the number of melody sequences generated so far in the melody sequence stream is less than or equal to a user-predefined sliding window size w. In this phase, each item in a new incoming melody sequence is transformed into its bit sequence representation.

MSID	Melody Sequence	Bit-Sequences in current SW_1
m_1	(acd)	$\mathbf{Bit}(a)$=100, $\mathbf{Bit}(c)$=100, $\mathbf{Bit}(d)$=100
m_2	(bce)	$\mathbf{Bit}(a)$=100, $\mathbf{Bit}(c)$=110, $\mathbf{Bit}(d)$=100, $\mathbf{Bit}(b)$=010, $\mathbf{Bit}(e)$=010
m_3	$(abce)$	$\mathbf{Bit}(a)$=101, $\mathbf{Bit}(c)$=111, $\mathbf{Bit}(d)$=100, $\mathbf{Bit}(b)$=011, $\mathbf{Bit}(e)$=011

Figure 3: Bit-sequence representations after window initialization phase

Example 3 Consider the melody sequence stream in Figure 2. The first sliding window SW_1 contains three melody sequences: m_1, m_2, and m_3. The bit-sequences of items of SW_1 in the window initialization phase are shown in Figure 3.

3.2.2. Window Sliding Phase

The phase is activated after the sliding window SW becomes full. A new incoming melody sequence is appended to the sliding window, and the oldest melody sequence is removed from the current window.

For removing oldest information, an efficient method is used in the proposed algorithm. Based on the bit-sequence representation, FTP-MQS uses the *bitwise left shift* operation to remove the aged melody sequences from the set of items in the current sliding window. After sliding the window, an effective pruning method, called *Item-Prune*, is used to improve the memory requirement of FTP-MQS. The pruning approach is that *a 1-itemset X in the current sliding window is dropped if and only if* $\mathbf{sup}(X) = 0$.

ID	Melody Sequences	Bit-Sequences
SW_1	<m_1, (acd)> <m_2, (bce)> <m_3, $(abce)$>	$\mathbf{Bit}(a)$ = 101, $\mathbf{Bit}(c)$ = 111, $\mathbf{Bit}(d)$ = 100, $\mathbf{Bit}(b)$ = 011, $\mathbf{Bit}(e)$ = 011
SW_2	<m_2, (bce)> <m_3, $(abce)$> <m_4, (be)>	$\mathbf{Bit}(a)$ = 010, $\mathbf{Bit}(c)$ = 110, ~~$\mathbf{Bit}(d)$ = 000~~, $\mathbf{Bit}(b)$ = 111, $\mathbf{Bit}(e)$ = 111

Figure 4: Bit-sequences of items after sliding SW_1 to SW_2

Example 4 Consider the melody sequence stream in Figure 2. Before the fourth melody sequence <m_4, (be)> is processed, the first melody sequence m_1 must be removed from the current window using bitwise left shift on the set of items. Hence, $\mathbf{Bit}(a)$ is modified from 101 to 010. Similarly, $\mathbf{Bit}(c)$, $\mathbf{Bit}(d)$, $\mathbf{Bit}(b)$ and $\mathbf{Bit}(e)$ are modified to 110, 000, 110, and 110, respectively. Then, the new melody sequence <m_4, (be)> is processed using bit-sequence transform. The result is shown in Figure 4. Note that item d is dropped since $\mathbf{Bit}(d)$ = 000, i.e., $\mathbf{sup}(d) = 0$.

3.2.3. Frequent Temporal Patterns Generation Phase

The phase is performed only when the up-to-date set of frequent temporal patterns is requested. In this phase, FTP-MQS uses a level-wise method to generate the set of candidate temporal patterns CTP_k (candidate temporal patterns with k items) from the pre-known frequent temporal patterns FTP_{k-1} (frequent temporal patterns with k-1 items) according to the *Apriori* property [1][1]. The step is called *CandidateGenApriori* (Candidate temporal pattern Generation using Apriori property). Then, the proposed algorithm uses the *bitwise AND* operation to compute the support of these candidates in order to find the frequent ones FTP_k. The generation-then-test process is stopped until no new candidates with k+1 items (CTP_{k+1}) are generated. FTP-MQS algorithm is shown in Figure 5.

Algorithm FTP-MQS

Input: *MSS* (a continuous melody sequence stream), *s* (a user-defined minimum support threshold), and *w* (the size of sliding window).

Output: a set of frequent temporal patterns, *FTP-Output*.

Begin

```
  SW = NULL;  /* SW consists of w melody sequences */
  Repeat:
     for each incoming melody sequence m_i in SW do
        if SW = FULL then
           Do bitwise-shift on bit-sequences of all items in SW;
        end if
        for each item X in m_i do
           Do bit-sequence transform(X);
        end for
     end for
     for each bit-sequence Bit(X) in SW do
        if sup(X) = 0 then
           Drop X from SW;
        end if
     end for
/* The following is the frequent temporal patterns generation phase. The phase is
performed only when the up-to-date set of frequent temporal patterns is requested. */
     FTP_1 = {frequent temporal 1-itemsets};
     for (k=2; FTP_{k−1} ≠ NULL; k++) do
        CTP_k = CandidateGenApriori(FTP_{k−1});
        Do bitwise AND to find the supports of CTP_k;
        for each candidate c_k ∈ CFP_k do
           if sup(c_k) ≥ w·s then
              FTP_k = {c_k ∈ CFP_k | sup(c_k) ≥ w·s};
           end if
        end for
     end for
  FTP-Output = ∪_k FTP_k;
```

End

Figure 5: Algorithm FTP-MQS

Example 5 Consider the bit-sequences of SW_2 in Figure 4, and let the minimum support threshold *s* be 0.6. Hence, a temporal pattern *X* is *frequent* if $\mathbf{sup}(X) \geq 0.6 \cdot 3 = 1.8$. In the following, we discuss the frequent temporal patterns mining steps of SW_2. The generated patterns are shown in Figure 2.

First, FTP-MQS generates candidate 2-itemsets, (*bc*), (*be*) and (*ce*), by combining frequent 1-itemsets: (*b*), (*c*) and (*e*), where **Bit**(*b*) = 111, i.e., **sup**(*b*) = 3, **Bit**(*c*) = 110, i.e., **sup**(*c*) = 2, and **Bit**(*e*) = 110, i.e., **sup**(*e*) = 2. 1-itemset (*a*) is an *infrequent* temporal pattern, since its **Bit**(*a*) = 010, i.e., **sup**(*a*) = 1. All these candidates are frequent temporal patterns after using bitwise AND operations to count the supports of these candidates. Because the **Bit**(*bc*) is 110, the support of candidate 2-itemset *bc* are 2, i.e., **sup**(*bc*) = 2. Similarity, **sup**(*be*) = 3, and **sup**(*ce*) = 2.

Second, FTP-MQS generates one candidate 3-itemset (*bce*) according to Apriori property and uses bitwise AND operation to count the **sup**(*bce*) = 2, i.e., **Bit**(*bc*) AND **Bit**(*be*) AND **Bit**(*ce*) = 110. Because no new candidates are generated, the generation-then-test process is stopped. Hence, there are six frequent temporal patterns, (*b*), (*c*), (*bc*), (*be*), (*ce*), (*bce*), generated by FTP-MQS in SW_2. The process is shown in Figure 6.

Melody Sequences (SW_2)	Bit-Sequences in SW_2
<m_2, (*bce*) >	**Bit**(*a*) = 010
<m_3, (*abce*) >	**Bit**(*c*) = 110
<m_4, (*be*) >	**Bit**(*b*) = 111
	Bit(*e*) = 111

FTP_1 in SW_2 (*s* = 0.6)	sup
{(*b*) | **Bit**(*b*) = 111}	3
{(*c*) | **Bit**(*c*) = 110}	2
{(*e*) | **Bit**(*e*) = 111}	3

CTP_2 in SW_2
{(*bc*) | **Bit**(*b*) = 111 AND **Bit**(*c*) = 110}
{(*be*) | **Bit**(*b*) = 111 AND **Bit**(*e*) = 111}
{(*ce*) | **Bit**(*c*) = 110 AND **Bit**(*e*) = 111}

FTP_2 in SW_2	sup
{(*bc*) | **Bit**(*bc*) = 110}	2
{(*be*) | **Bit**(*be*) = 111}	3
{(*ce*) | **Bit**(*ce*) = 110}	2

CTP_3 in SW_2
{(*bce*) | **Bit**(*bc*) = 110 AND **Bit**(*be*) = 111 AND **Bit**(*ce*) = 110}

FTP_3 in SW_2	sup
{(*bce*) | **Bit**(*bce*) = 110}	2

Figure 6: Steps of frequent temporal patterns generation in SW_2

4. EXPERIMENTS

In this section, we report the experimental results of the proposed algorithm FTP-MQS. All the programs are implemented using Microsoft Visual C++ Version 6.0 and performed on a 1.80 GHz Pentium(R) PC machine with 512 MB memory running on Windows 2000. For testing frequent temporal patterns mining of melody sequence streams, we generate melody sequence streams using IBM synthetic data generator proposed by Agrawal and Srikant [1]. One synthetic melody sequence stream, denoted by T5.I4.D1000K, of size 1 million melody sequences each are used to evaluate the performance of the proposed algorithm FTP-MQS. T5.I4.D1000K, with 1,000 unique items, has an average melody sequence size of 5 items with average maximal frequent temporal pattern size of 4 items. In all experiments, the melody sequences are looked up in sequence to simulate the environment of an online data stream.

The experiments of memory requirements are shown in Figures 7 and 8, and the processing times are shown in Figures 9, 10, and 11. The minimum support threshold *s* and the size of a window *w* are set to 0.1% and 20,000, respectively. Figure 7 shows the memory usage of the window initialization phase and window sliding phase. As shown in Figure 7, FTP-MQS consumes only about 2.1MB in window initialization phase, but the memory consumption of original data is increased linearly from 0.2MB to 3.9MB. In window sliding phase, the memory usage of FTP-MQS is approximately a half of original data. Figure 8 shows the memory usage of the frequent temporal patterns generation phase. In frequent temporal patterns generation phase, the memory requirement of FTP-MQS is between 33.5MB to 39MB.

[1] It is a **downward closure property**, i.e., *if a temporal pattern is frequent, all of its sub-patterns will also be frequent*.

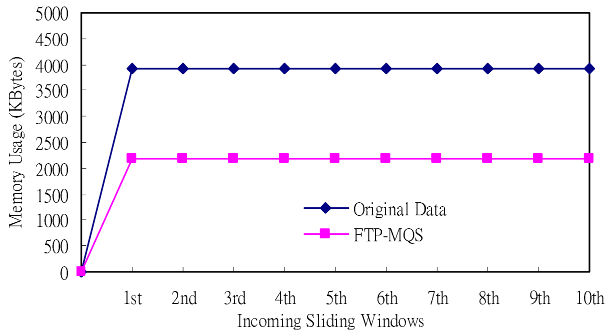

Figure 7: Memory usages in the window initialization phase and window sliding phase.

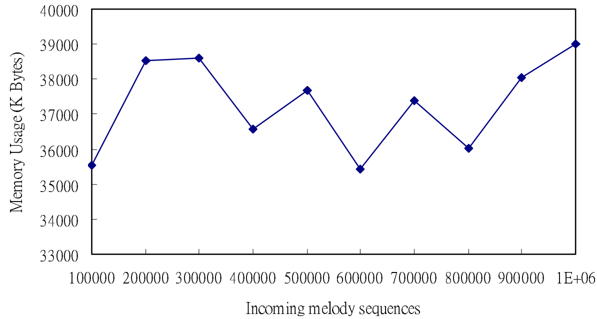

Figure 8: Memory usage in frequent temporal patterns generation phase

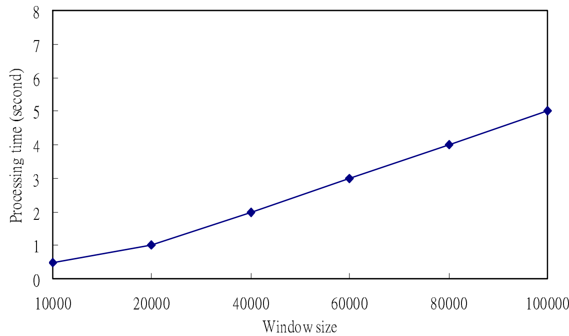

Figure 9: Processing time of window initialization phase under different window sizes

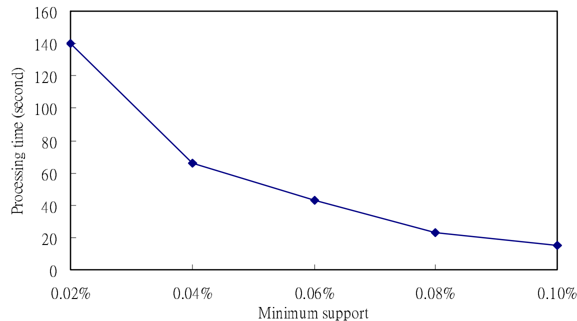

Figure 10: Mining time of frequent temporal patterns under different minimum supports

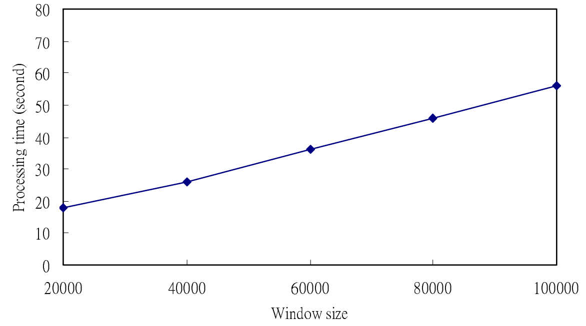

Figure 11: Mining time of frequent temporal patterns under different window sizes

Figure 9 shows the processing time of window initialization phase of FTP-MQS under different window sizes from 10,000 melody sequences to 100,000 melody sequences. Figure 10 shows the mining time of frequent temporal patterns using various minimum support thresholds from 0.02% to 0.1%. Figure 11 shows the mining time of frequent temporal patterns under different window sizes from 20,000 melody sequences to 100,000 melody sequences.

5. CONCLUSIONS

In this paper, we study the problem of mining frequent temporal patterns from a continuous music query stream with a sliding window. A new online algorithm, called FTP-MQS, is proposed. An effective bit-sequence representation is developed to maintain the essential information of recent frequent temporal patterns. Experiments show that the proposed algorithm is an efficient single-pass algorithm for mining music query streams.

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] M. M. Gaber, A. Zaslavsky and S. Krishnaswamy, "Mining data streams: a review," ACM SIGMOD Record, 34(1), June 2005.

[5] 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.

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

[7] H.-F. Li, S.-Y. Lee and M.-K. Shan, "Mining frequent closed structures in streaming melody sequences," in: Proc. ICME, 2004.

[8] 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.

[9] 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.

[10] 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.

[11] www.music.com.tw

[12] www.kkbox.com.tw