Available online at www.sciencedirect.com

SCIENCE DIRECT®

Pattern Recognition Letters 26 (2005) 1658–1674

Pattern Recognition Letters

www.elsevier.com/locate/patrec

Online mining maximal frequent structures in continuous landmark melody streams

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

[a] *Department of Computer Science and Information Engineering, National Chiao-Tung University, 1001 Ta Hsueh Road, Hsin-Chu 300, Taiwan*

[b] *Department of Computer Science, National Chengchi University, 64, Sec. 2, Zhi-nan Road, Wenshan, Taipei 116, Taiwan*

Received 10 January 2004; received in revised form 13 November 2004
Available online 14 April 2005
Communicated by E. Backer

Abstract

In this paper, we address the problem of online mining maximal frequent structures (*Type I & II melody structures*) in unbounded, continuous landmark melody streams. An efficient algorithm, called MMS_{LMS} (Maximal Melody Structures of Landmark Melody Streams), is developed for online incremental mining of maximal frequent melody substructures in *one scan* of the continuous melody streams. In MMS_{LMS}, a space-efficient scheme, called CMB (Chord-set Memory Border), is proposed to constrain the upper-bound of space requirement of maximal frequent melody structures in such a streaming environment. Theoretical analysis and experimental study show that our algorithm is efficient and scalable for mining the set of all maximal melody structures in a landmark melody stream.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Machine learning; Data mining; Landmark melody stream; Maximal melody structure; Online algorithm

1. Introduction

Recently, database and knowledge discovery communities have focused on a new data model, where data arrives in the form of continuous, rapid, huge, unbounded *streams*. It is often referred to as 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 record and click streams in Web applications, performance measurement in network monitoring and traffic management, call records in telecommunications, etc. In such a data

[*] Corresponding author. Tel.: +886 35731901; fax: +886 35724176.

E-mail addresses: hfli@csie.nctu.edu.tw (H.-F. Li), sylee@csie.nctu.edu.tw (S.-Y. Lee), mkshan@cs.nccu.edu.tw (M.-K. Shan).

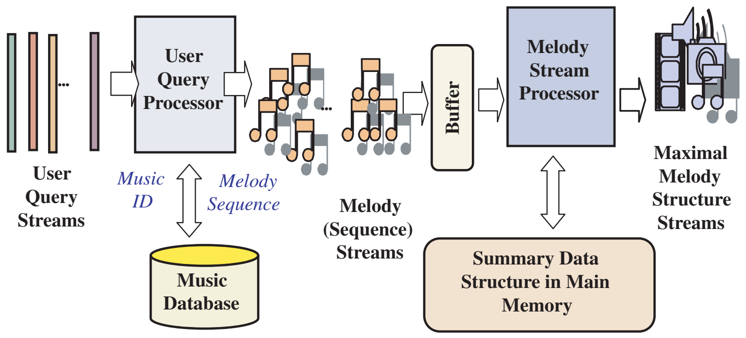

Fig. 1. Computation model for music melody streams.

stream model, knowledge discovery has two major characteristics (Babcock et al., 2002). First, the volume of a continuous stream over its lifetime could be huge and fast changing. Second, the continuous queries (not just one-shot 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 of in-memory *summary data structure* with small memory footprints that can support both one-time and continuous queries. In other words, data stream mining algorithms have to sacrifice the exactness of its analysis result by allowing some counting error.

Although several techniques have been developed recently for discovering and analyzing the content of *static music* data (Bakhmutora et al., 1997; Hsu et al., 2001; Shan and Kuo, 2003; Yoshitaka and Ichikawa, 1999; Zhu et al., 2001), new techniques are needed to analyze and discover the content of *streaming* music data. Thus, this paper studies a new problem of how to discover the maximal melody structures in a continuous unbounded melody stream. The problem comes from the context of online music-downloading services (such as *Kuro* at www.music.com.tw), 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 maximal 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 the music melody 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 retrieving the music database. Note that a buffer can be optionally set for temporary storage of recent music melodies from the music melody streams.

In this paper, we present a novel algorithm $\mathrm{MMS}_{\mathrm{LMS}}$ (Maximal Melody Structures of Landmark Melody Streams) for mining the set of all maximal melody structures in a landmark melody stream. Moreover, the music melody data and patterns are represented as sets of chord-sets (*Type I Melody structures*) or strings of chord-sets (*Type II Melody structures*). While providing a general framework of music stream mining, algorithm $\mathrm{MMS}_{\mathrm{LMS}}$ has two major features, namely *one scan of music melody streams for online frequency collection*, and *prefix-tree-based compact pattern representation*. With these two important features, $\mathrm{MMS}_{\mathrm{LMS}}$ is provided with the capability to work continuously in the unbounded streams for an arbitrary long time with bounded resources, and to quickly answer users' queries at any time.

2. Preliminaries

2.1. Music terminologies

In this section, we describe several features of music data used in this paper. For the basic terminologies on music, we refer to (Jones, 1974). A *chord* is a 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 (Shan and Kuo, 2003).

Definition 1. The *type I melody structure* is represented as a *set* of chord-sets. The *type II melody structure* is represented as a *string* of chord-sets.

2.2. Problem statement

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 $S = \langle x_1 x_2 \cdots x_m \rangle$, where $x_i \in \Psi$, $\forall i = 1,2,\ldots,m$. A *block* is a set of melody sequences.

Definition 2. A *landmark melody stream LMS* = $[B_1, B_2, \ldots, B_N)$, is an *infinite* sequence of blocks, where each block B_i is associated with a *block identifier i*, and N is the identifier of the "*latest*" block B_N. The *current length* of LMS, written as |LMS|, is N. The blocks arrive in some order (implicitly by *arrival time* or explicitly by *time-stamp*), and may be seen only *once*.

Definition 3. A set $Y \subseteq \Psi$ is called an *item-set*, i.e., *a set of chord-sets*. *k-item-set* is represented by $(y_1 y_2 \cdots y_k)$. The *support* of an item-set Y, denoted as $\sigma(Y)$, is the number of melody sequences containing Y as a *subset* in the LMS seen so far. An item-set is *frequent* if its support is greater than or equal to $minsup \cdot |\mathrm{LMS}|$, where *minsup* is a user-specified minimum support threshold in the range of [0, 1], and |LMS| is the current length of the landmark melody stream LMS.

Definition 4. A string Z is called an *item-string*, i.e., *a string of chord-sets*. A *k-item-string* is represented by $\langle z_1 z_2 \cdots z_k \rangle$, where $z_i \in \Psi$, $\forall i = 1,2,\ldots,k$. The *support* of an item-string Z, denoted as $\sigma(Z)$, is the number of melody sequences containing Z as a *substring* in the LMS seen so far. An item-string is *frequent* if its support is greater than or equal to $minsup \cdot |\mathrm{LMS}|$, where *minsup* is a user-specified minimum support threshold in the range of [0, 1], and |LMS| is the current length of the landmark melody stream seen so far.

Definition 5. A frequent item-set (or item-string) is called *maximal* if it is not a subset (or substring) of any other frequent item-set (or item-string).

In fact, the total number of maximal melody structures is smaller than that of frequent melody structure. Hence, the type of maximal melody structures is more suitable for the performance requirements of music stream mining.

Definition 6. (Problem Definition of Online Mining Maximal Melody Structures in Continuous Landmark Melody Streams.) Given a landmark melody stream LMS = $[B_1, B_2, \ldots, B_N)$ and the user specified minimum support, *minsup*, in the range of [0, 1], the problem of online mining maximal melody substructures is to discover the set of all maximal melody structures, i.e., maximal item-sets or maximal item-strings, in single one scan of the landmark music stream.

2.3. Main performance requirements of music melody stream mining

The main performance challenges of mining melody streams are:

(1) *Online*, *one-pass algorithm*: each sequence in the landmark melody stream is examined once.
(2) *Bounded-storage*: limited memory for storing crucial, compressed information in summary data structure.
(3) *Real-time*: per item processing time must be low.

The proposed $\mathrm{MMS}_{\mathrm{LMS}}$ algorithm possesses all of these characteristics, while none of previously published methods (Bakhmutora et al., 1997; Hsu et al., 2001; Shan and Kuo, 2003; Yoshitaka and Ichikawa, 1999; Zhu et al., 2001) can claim the same.

3. Online mining maximal frequent structures in landmark melody streams

3.1. Chord-set memory border

In this section, the upper bound on the number of candidate maximal melody structures is discussed, and an efficient algorithm for chord-set memory border construction is proposed.

Theorem 1. *Given a set of k frequent chord-sets from a landmark melody stream, an upper bound of the amount of maximal frequent melody structures is* $C^k_{\lceil k/2 \rceil}$.

Proof. Assume that there are k frequent chord-sets, i.e., k frequent items, in the current landmark melody stream. The solution space of mining all frequent item-sets in the worst case is $C^k_1 + C^k_2 + \cdots + C^k_i + \cdots + C^k_{\lceil k/2 \rceil} + \cdots + C^k_k$, where C^k_1 is the total number of frequent 1-item-sets, C^k_i is that of frequent i-item-sets, and C^k_k is that of frequent k-item-sets. We observe that the value of $C^k_{\lceil k/2 \rceil}$ is the maximum value among all the binominal coefficient C^k_i, $\forall i = 1, 2, \ldots, k$, in mining all frequent i-item-sets. In other words, the number of frequent $\lceil k/2 \rceil$-item-sets is a maximum. We will prove the number of maximal frequent item-sets can not be greater than the value $C^k_{\lceil k/2 \rceil}$, i.e., $C^k_{\lceil k/2 \rceil}$ is the upper bound. We prove it by contradiction.

Assume that the value of $C^k_{\lceil k/2 \rceil}$ is not the maximum number of maximal frequent item-sets, i.e., a *larger* upper bound U exists, where $U > C^k_{\lceil k/2 \rceil}$. Consider that there are one or more frequent melody structures with length L, where $L > \lceil k/2 \rceil$. If F is a frequent melody structure with length $\lceil k/2 \rceil + i$ and it is maximal, where $i = 1,2, \ldots, k - \lceil k/2 \rceil$, then all of the substructures of F are frequent, which is based on the *anti-monotone Apriori heuristic* (Agrawal and Srikant, 1994): *if any i-item-set* (*or i-item-string*) *is not frequent*, *its* ($i + 1$)-item-set (or ($i + 1$)-item-string) can never be frequent, but not maximal, which is based on the definition 5: *a frequent item-set* (*or item-string*) *is called maximal if it is not a subset* (*or substring*) *of any other frequent item-set* (*or item-string*). In other words, it means that when *one* maximal frequent structure with length L, where $L > \lceil k/2 \rceil$, is *added*, at most L frequent melody structures with length $L - 1$, are *decremented* from the current collection of maximal frequent melody structures found so far. Hence, the maximum number of maximal melody structures is changed from U to U', where $U' = U + 1 - L$, which is not greater than $C^k_{\lceil k/2 \rceil}$. This conflicts with the assumption of $U > C^k_{\lceil k/2 \rceil}$ and results in a contradiction. Thus the statement is proven to be true. Therefore, we conclude that the maximum number of maximal melody structures is $C^k_{\lceil k/2 \rceil}$ in the problem of online mining maximal melody structures in a landmark melody stream. □

Example 1. Assume that there are five frequent items (i.e., frequent 1-item-sets) a, b, c, d, and e in the landmark melody stream as shown in Fig. 2. Let MF denote the total number of maximal frequent item-sets. At this point, a, b, c, d and e are maximal and $MF = C^5_1$. Based on the Apriori heuristic, C^5_2 frequent 2-item-sets are discovered in the worst case. In this case, these frequent 2-item-sets are also maximal and those frequent 1-item-sets are not maximal any more. The current MF is $C^5_1 + C^5_2 - C^5_1 = C^5_2$. Next, C^5_3 frequent 3-item-sets are found in the worst case. These frequent 3-item-sets are maximal but the sub-sets of the maximal 3-item-sets, i.e., frequent 2-item-sets, are not maximal any more. Now, the MF becomes $C^5_2 + C^5_3 - C^5_2 = C^5_3$. At this time, suppose the frequent 4-item-set *abcd* exists in this instance and it is also a maximal 4-item-set. The frequent subsets, with length three, of *abcd*, i.e., *abc*, *abd*, *acd* and *bcd*, are not maximal any more. Now, the MF becomes $C^5_3 + 1 - 4 = 7$, i.e., *abcd*, *ace*, *ade*, *bcd*, *bce*, *bde*, *cde* are maximal frequent item-sets. The new MF is smaller than the upper bound $C^5_{\lceil 5/2 \rceil}$. Hence, we can find that if one or more frequent item-sets with length L, where $L > \lceil 5/2 \rceil$, are added into the collection of maximal frequent item-sets found so far, the value of MF would be changed and would be less than $C^5_{\lceil 5/2 \rceil}$. Consequently, the $C^5_{\lceil 5/2 \rceil}$ is the upper bound of the number of maximal melody structures.

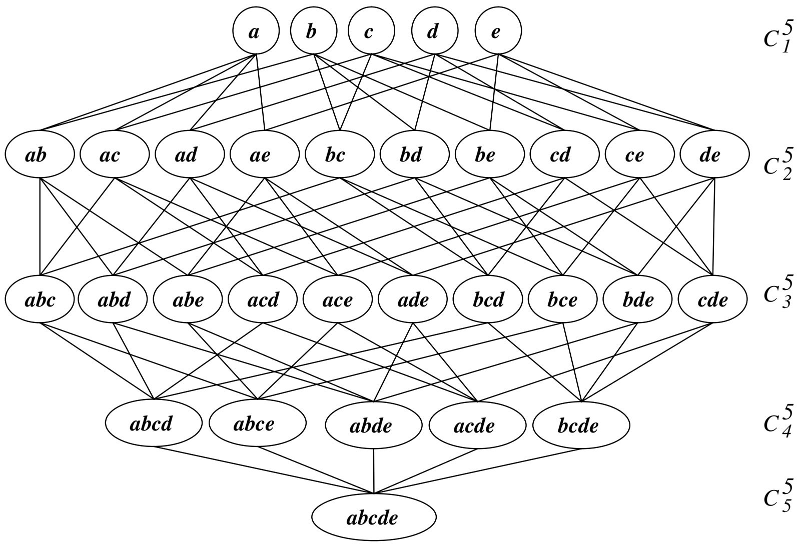

Fig. 2. Item-set enumeration lattice with respect to five items: *a*, *b*, *c*, *d* and *e*.

The key property of algorithm MMS_{LMS} is derived from the recent work (Karp et al., 2003) for finding frequent elements in streaming data. The basic scheme of mining chord-sets from music data streams is generalized from the well-known algorithm (Fisher and Salzberg, 1982) for determining whether a value (*majority element*) occurs more than $n/2$ times, i.e., $minsup = 0.5$, in a data stream of length n.

The method can be extended to an arbitrary value of *minsup*. The scheme is processed as follows. At any given time, a superset of k probably frequent chord-sets with at most $1/minsup$ times is maintained. Initially, the set is empty. As a chord-set is read from the melody sequence in the current block, two operations are performed as follows. First, if the current chord-set is not contained in the superset and some entries are free, it is inserted into the superset with a count set to one. Second, if the chord-set is already in the superset, its count is incremented by one. However, if the superset is full, the count of each entry in the superset is decremented by one, and the chord-sets whose frequencies are just one are dropped. The method thus identifies at most k candidates for having appeared more than $n/(k+1)$ times, and uses $O(1/minsup)$ memory entries.

3.2. The proposed algorithm: MMS_{LMS}

Algorithm MMS_{LMS} has three modules: MMS_{LMS}-*buffer*, MMS_{LMS}-*summary*, and MMS_{LMS}-*mine*. MMS_{LMS}-buffer repeatedly reads in a block of melody sequences into available main memory. All compressed and essential information about the maximal melody structures will be maintained in the MMS_{LMS}-summary. Finally, MMS_{LMS}-mine finds the maximal melody structures by a depth-first manner in the current MMS_{LMS}-summary. Therefore, the challenges of online mining landmark melody streams lie in the design of a *space-efficient* representation of the in-memory summary data structure and a *fast* discovery algorithm for finding maximal melody structures in real time.

3.2.1. MMS_{LMS}-summary

First of all, the in-memory data structure MMS_{LMS}-summary is defined and the constructing process of MMS_{LMS}-summary is discussed. Then we use a running example to illustrate.

Definition 7. A MMS_{LMS}-*summary* is an extended prefix-tree-based summary data structure defined below.

1. $\mathrm{MMS_{LMS}}$-*summary* consists of a CMB (*Chord-set Memory Border*), and a set of *MPI-trees* (*Maximal Prefix-Item trees of item-suffixes*) denoted as *MPI-trees*(*item-suffixes*).
2. Each node in the *MPI-tree*(*item-suffix*) consists of four fields: *item-id*, *support*, *block-id* and *node-link*, where *item-id* is the item identifier of the inserting item, *support* registers the number of melody sequences represented by a portion of the path reaching the node with the item-id, the value of *block-id* assigned to a new node is the block identifier of the current block, and *node-link* links up a node with the next node with the same item-id in the same MPI-tree or null if there is none.
3. Each entry in the CMB consists of four fields: *item-id*, *support*, *block-id*, and *head of node-link* (a pointer links to the root node of the MPI-tree with the same item-id), abbreviated as *head-link*, where *item-id* registers which item identifier the entry represents, *support* records the number of transactions containing the item carrying the item-id, the value of *block-id* assigned to a new entry is the block identifier of current block, and *head-link* points to the root node of the *MPI-tree*(*item-suffix*). Notice that each entry with *item-id* in the CMB is an *item-suffix* and it is also the *root node* of the MPI-tree(*item-id*).
4. Each *MPI-tree*(*item-suffix*) has a specific *CMB-table* (Chord-set Memory Border table) with respect to the item-suffix (denoted as *CMB-table*(*item-suffix*)). The *CMB-table*(*item-suffix*) is composed of four fields, namely *item-id*, *support*, *block-id*, and *head-link*. The *CMB-table*(*item-suffix*) operates the same as the CMB except that the field *head-link* links to the first node carrying the item-id in the *MPI-tree*(*item-suffix*). Notice that |CMB-*table*(*item-suffix*)| = |CMB| in the worst case, where |CMB| denotes the total number of entries in the CMB.

The construction of $\mathrm{MMS_{LMS}}$-summary is described as follows. First of all, $\mathrm{MMS_{LMS}}$ reads a melody sequence S from the current block. Then, $\mathrm{MMS_{LMS}}$ projects the sequence S into many subsequences and inserts these subsequences into the CMB and MPI-trees. In details, each melody sequence S, such as $\langle x_1,x_2,\ldots,x_m\rangle$, in the current block should be projected by inserting m *item-suffix melody subsequences* into the $\mathrm{MMS_{LMS}}$-summary. In other words, the melody sequence $S = \langle x_1,x_2,\ldots,x_m\rangle$ is converted into m melody subsequences; that is, $\langle x_1,x_2,\ldots,x_m\rangle$, $\langle x_2,x_3,\ldots,x_m\rangle$, $\ldots,\langle x_{m-1},x_m\rangle$, and $\langle x_m\rangle$. The m melody subsequences are called *item-suffix sequences*, since the first item of each melody subsequence is an *item-suffix* of the original melody sequence S. This step is called *sequence projection*, and is denoted as *Sequence-Projection* $(S) = \{x_1|S,x_2|S,\ldots,x_i|S,\ldots,x_m|S\}$, where $x_i|S = \langle x_i,x_{i+1},\ldots,x_m\rangle$, $\forall i = 1,2,\ldots,m$. Furthermore, the cost of sequence projection of a melody sequence with length m is $(m^2 + m)/2$, i.e., $m + (m - 1) + \cdots + 2 + 1$.

After *Sequence-Projection* (S), $\mathrm{MMS_{LMS}}$ algorithm removes the original melody sequence S from the $\mathrm{MMS_{LMS}}$-buffer. Next, the set of items in these item-suffix sequences are inserted into the CMB and the *MPI-trees*(*item-suffixes*) as a branch, and the *CMB-table*(*item-suffixes*) are updated according to the *item-suffixes*. If an item-set (or item-string) share a prefix with an item-set (or item-string) already in the tree, the new item-set (or item-string) will share a prefix of the branch representing that item-set (or item-string). In addition, a support counter is associated with each node in the tree. The counter is updated when an item-suffix sequence causes the insertion of a new branch.

In order to limit the memory size of the summary data structure $\mathrm{MMS_{LMS}}$-summary, a space pruning technique is performed. Let the minimum support threshold be *minsup*, in the range of [0, 1], and the current length of the landmark melody stream be N. The rule for space pruning is as follows. A melody structure E is deleted if $E.support < minsup \cdot N$. E is called an *infrequent* melody structure. After pruning all infrequent melody structures from the CMB, *CMB-table*-(*item-suffix*) and *MPI-trees*, the $\mathrm{MMS_{LMS}}$-summary contains all information about frequent melody structures of the landmark melody stream generated so far. Example 2 below illustrates the algorithm step by step. Note that the $\langle\ \rangle$ of sequences are omitted for clear presentation.

Example 2. Let a block B_j of the landmark melody stream LMS be $\langle acdef\rangle$, $\langle abe\rangle$, $\langle cef\rangle$, $\langle acdf\rangle$, $\langle cef\rangle$ and $\langle df\rangle$, and the minimum support threshold be 0.5 (i.e., *minsup* = 0.5), where *a*, *b*, *c*, *d*, *e* and *f* are chord-sets (i.e., *items*) in a landmark melody stream seen so far. MMS_{LMS} algorithm constructs the MMS_{LMS}-summary with respect to the incoming block B_j and prunes all item-sets that are infrequent from the current MMS_{LMS}-summary in the following steps. Note that each node or entry represented as (f_1:f_2:f_3) is composed of three fields: *item-id*, *support*, and *block-id*. For example, (*a*:2:*j*) indicates that, from block B_j, item *a* appeared twice.

Step 1: MMS_{LMS} reads current block B_j into main memory for constructing the MMS_{LMS}-summary.

(a) *First melody sequence acdef*: First of all, MMS_{LMS} algorithm reads the first melody sequence *acdef* and calls the *Sequence-Projection* (*acdef*). Then MMS_{LMS} inserts the item-suffix sequences *acdef*, *cdef*, *def*, *ef*, and *f* into the CMB, [MPI-tree(*a*), CMB-table(*a*)], [MPI-tree(*c*), CMB-table(*c*)], [MPI-tree(*d*), CMB-table(*d*)], [MPI-tree(*e*), CMB-table(*e*)], and [MPI-tree(*f*), CMB-table(*f*)], respectively. The result is shown in Fig. 3. In the following sub-steps, as demonstrated in Fig. 4 through Fig. 9, the *head-links* of each *CMB-table* (*item-suffix*) are omitted for concise presentation.

(b) *Second melody sequence abe*: MMS_{LMS} reads the second melody sequence *abe* and calls the *Sequence-Projection* (*abe*). Next, MMS_{LMS} inserts the item-suffix sequences *abe*, *be* and *e* into the CMB, [MPI-tree(*a*), CMB-table(*a*)], [MPI-tree(*b*), CMB-table(*b*)] and [MPI-tree(*e*), CMB-table(*e*)], respectively. The result is shown in Fig. 4.

(c) *Third melody sequence cef*: MMS_{LMS} reads the third melody sequence *cef* and calls the *Sequence-Projection* (*cef*). Then, MMS_{LMS} inserts the item-suffix sequences *cef*, *ef* and *f* into the CMB,

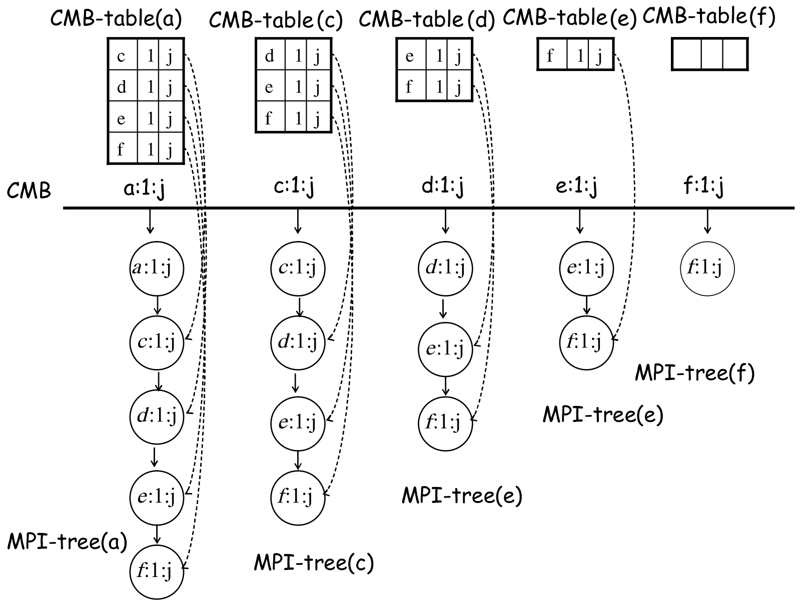

Fig. 3. MMS_{LMS}-summary construction after inserting first melody sequence *acdef* in block B_j. In the following sub-steps, as demonstrated in Fig. 4 through Fig. 9, the *head-links* of each *CMB-table* (*item-suffix*) are omitted for concise presentation.

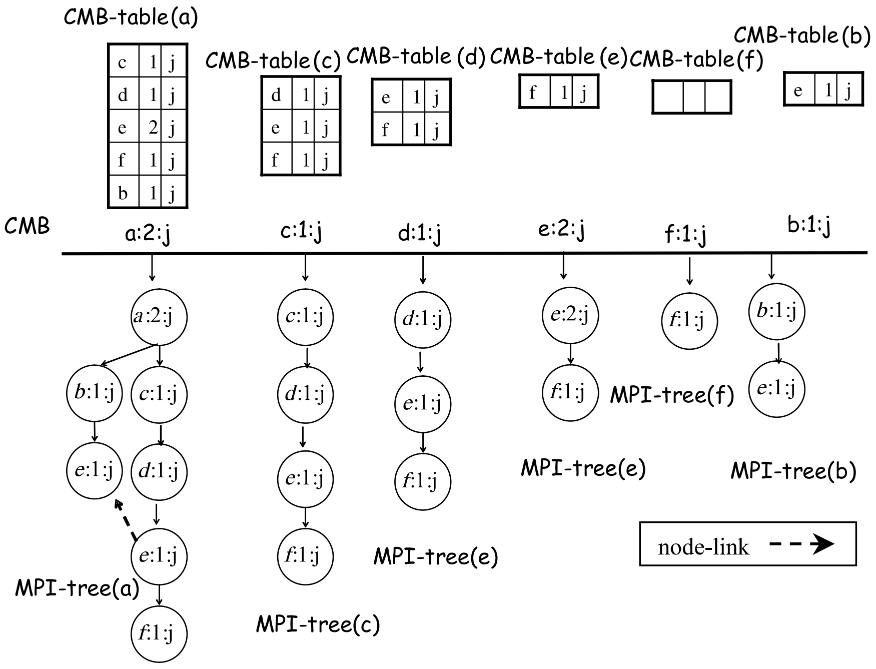

Fig. 4. MMS_{LMS}-summary construction after inserting second melody sequence *abe*.

[MPI-tree(*c*), CMB-table(*c*)], [MPI-tree(*e*), CMB-table(*e*)] and [MPI-tree(*f*), CMB-table(*f*)], respectively. The result is shown in Fig. 5.

(d) *Fourth melody sequence acdf*: MMS_{LMS} reads the fourth melody sequence *acdf* and calls the *Sequence-Projection* (*acdf*). Next, MMS_{LMS} inserts the item-suffix sequences *acdf*, *cdf*, *df* and *f* into the CMB, [MPI-tree(*a*), CMB-table(*a*)], [MPI-tree(*c*), CMB-table(*c*)], [MPI-tree(*d*), CMB-table(*d*)] and [MPI-tree(*f*), CMB-table(*f*)], respectively. The result is shown in Fig. 6.

(e) *Fifth melody sequence cef*: MMS_{LMS} reads the fifth melody sequence *cef* and calls the *Sequence-Projection* (*cef*). Then, MMS_{LMS} inserts the item-suffix sequences *cef*, *ef* and *f* into the CMB, [MPI-tree(*c*), CMB-table(*c*)], [MPI-tree(*e*), CMB-table(*e*)] and [MPI-tree(*f*), CMB-table(*f*)], respectively. The result is shown in Fig. 7.

(f) *Sixth melody sequence df*: MMS_{LMS} reads the sixth melody sequence *df* and calls the *Sequence-Projection* (*df*). Next, MMS_{LMS} inserts the item-suffix sequences *df* and *f* into the CMB, [MPI-tree(*d*), CMB-table(*d*)] and [MPI-tree(*f*), CMB-table(*f*)], respectively. The result is shown in Fig. 8.

Step 2: After computing the current block B_j, all infrequent melody structures are pruned by MMS_{LMS} from the current MMS_{LMS}-summary. At this time, MMS_{LMS} deletes the MPI-tree(*b*) and its corresponding CMB-table(*b*), and prunes the entry *b* from the CMB, since item *b* is an *infrequent* item; that is, $\sigma(b) < minsup \cdot |\text{LMS}|$, where $\sigma(b) = 1$ and $minsup \cdot |\text{LMS}| = 0.5 \cdot 6 = 3$. Next, MMS_{LMS} reconstructs the MPI-tree(*a*) by eliminating the information about the infrequent item *b*. The result is shown in Fig. 9.

The description stated above is the constructing process of MMS_{LMS}-summary with respect to the

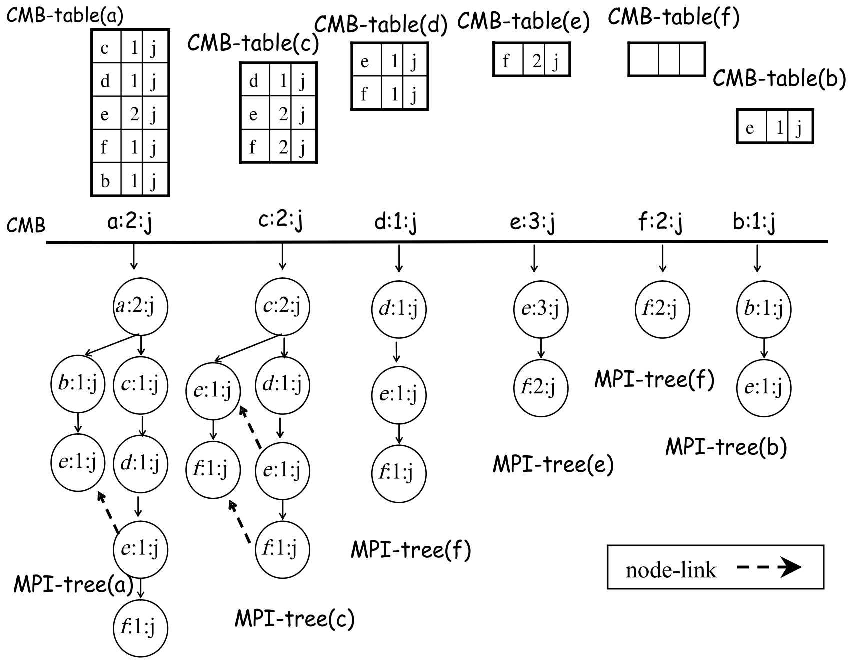

Fig. 5. MMS_{LMS}-summary construction after inserting third melody sequence *cef*.

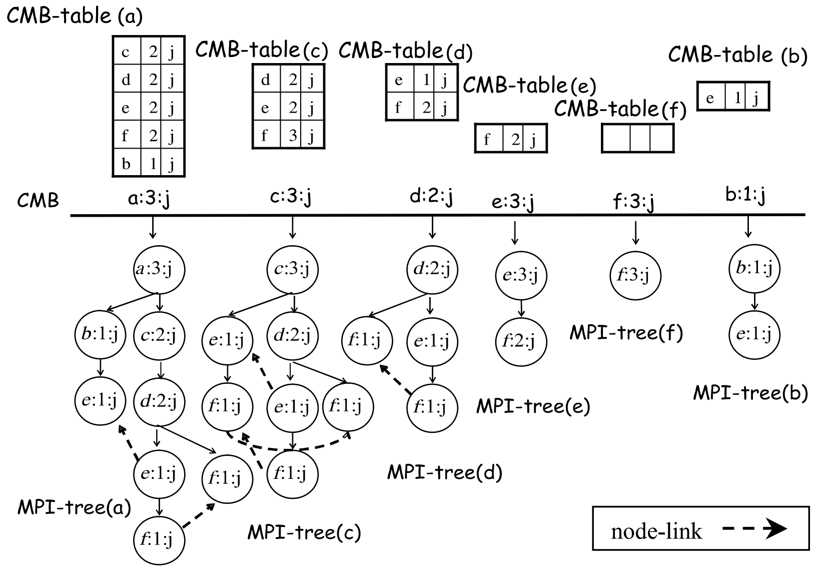

Fig. 6. MMS_{LMS}-summary construction after inserting third melody sequence *acdf*.

incoming block over a landmark melody stream. The MMS_{LMS}-summary construction algorithm is depicted in Fig. 10.

3.2.2. MMS_{LMS}-*mine*

In this section, the module, called MMS_{LMS}-*mine*, of mining maximal melody item-sets and

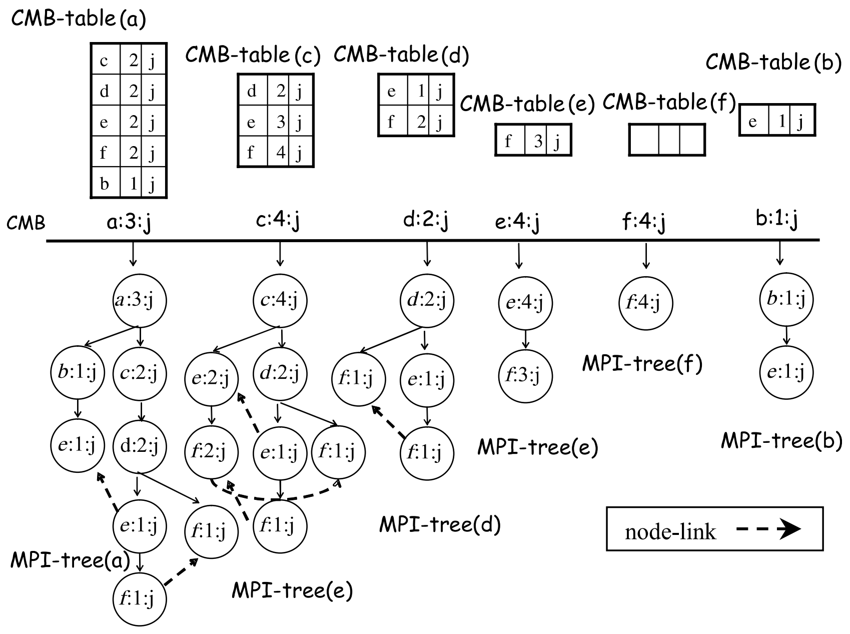

Fig. 7. MMS_{LMS}-summary construction after inserting fifth melody sequence *cef*.

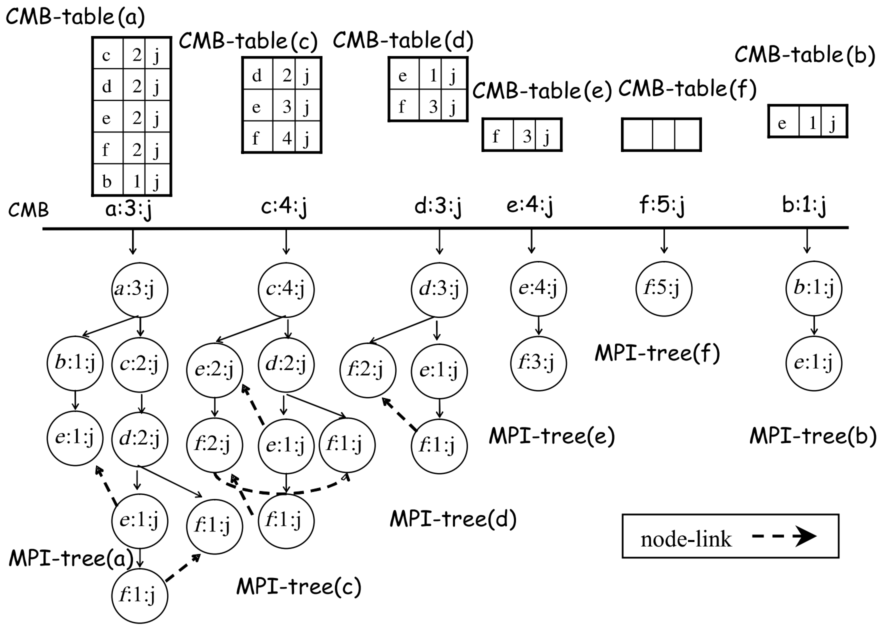

Fig. 8. MMS_{LMS}-summary construction after inserting sixth melody sequence *df*.

maximal melody item-strings from the current MMS_{LMS}-summary is discussed (Fig. 11).

First of all, given an entry *id* (from left to right, for example) in the current CMB, MMS_{LMS}-mine

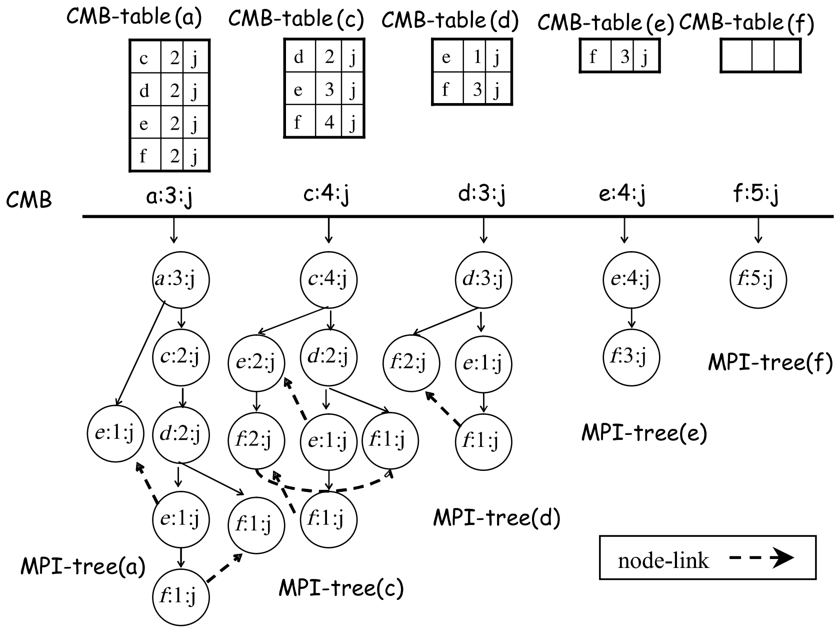

Fig. 9. Current MMS_{LMS}-summary after pruning all infrequent melody structures with respect to infrequent item *b*.

generates candidate maximal melody structures by a top-down approach. The top-down method uses the frequent items (i.e., chord-sets) of CMB-table(*id*) and item *id* to generate the candidates. The generating order of these candidates is determined by the size of item-set, from item-set size $1 + |\text{CMB-table}(id)|$ down to size 2. Note that, the generating order ends in 2-item-sets because all frequent entries in the current CMB-table are frequent 1-item-sets. Then MMS_{LMS}-mine checks these candidates whether they are frequent or not by traversing the MPI-tree(*id*). The MPI-tree traversing principle is described as follows. First, MMS_{LMS}-mine generates a candidate maximal melody item-set, $(j + 1)$-item-set, containing the item *id* and all items of the CMB-table(*id*), where $|\text{CMB-table}(id)| = j$. Second, MMS_{LMS}-mine traverses the MPI-tree via the node-links of the frequent candidate. After that if the candidate is not a frequent item-set, MMS_{LMS}-mine generates substructure candidates with *j*-item-sets. Next, MMS_{LMS}-mine executes the same MPI-tree traversing scheme for item-set counting. The process stops when MMS_{LMS}-mine finds all maximal frequent melody structures from the current MMS_{LMS}-summary. Moreover, MMS_{LMS}-mine stores these maximal melody structures into a temporal pattern list, called MMS_{LMS}-*list*. Notice that MMS_{LMS}-mine can find the set of frequent 2-item-sets by combining the item-suffix *id* with the frequent items of the CMB-table(*id*).

Example 3. This example illustrates the mining of the maximal melody item-sets from the current MMS_{LMS}-summary in Fig. 9. Let the minimum support threshold be 0.5, i.e., $minsup = 0.5$.

(1) Now, we start the maximal melody item-set mining scheme from the frequent item *a*. At this moment, the frequent item-set is the only 1-item-set (*a*), since the support of items *c*, *d*, *e* and *f* in the CMB-table (*a*) are less than $minsup \cdot |\text{LMS}|$, where $|\text{LMS}| = |B_j| = 6$.
(2) Next, MMS_{LMS}-mine starts on the second entry *c* for maximal melody item-set mining. MMS_{LMS}-mine generates a candidate maximal 3-item-set (*cef*), and traverses the MPI-tree(*c*) for counting its support. As a result,

```
Algorithm 1 (MMS_LMS-summary Construction)
Input: A landmark melody stream, $LMS = [B_1, B_2, \ldots, B_N)$, and a user-specified minimum
       support threshold, minsup.
Output: A current MMS_LMS-summary.

1:  CMB = $\varnothing$ /*initialize the CMB to empty.*/
2:  foreach block $B_j$ do /* $j = 1, 2, \ldots, N$ */
3:    foreach melody sequence $S = <x_1, x_2, \ldots, x_m> \in B_j$ ($j = 1, 2, \ldots, N$) do
4:      foreach item $x_i$   S do /*the CMB maintenance*/
5:        if $x_i \notin$ CMB then
6:          create a new entry ($x_i$, 1, j, head-link) into the CMB;
            /* the entry form is (item-id, support, block-id, head-link)*/
7:        else /* the entry already in the CMB*/
8:          $x_i$.support = $x_i$.support + 1; /* increment the support of item-id $x_i$ by one*/
9:        end if
10:     end for
11:     call Sequence-Projection(S);
        /* project the sequence with every prefix-item $x_i$ for the construction of MPI-tree($x_i$)*/
12:   end for
13:   call MMS_LMS-summary-pruning(MMS_LMS-summary, minsup, |LMS|);
14: end for

Subroutine Sequence-Projection
Input: A melody sequence $S = <x_1, x_2, \ldots, x_m>$ and the current block-id j;
Output: MPI-trees($x_i$), $\forall i$=1, 2, …, m;

1:  foreach item $x_i$ ($i$ =1, 2, …, m) do
2:    MPI-tree-maintenance([$x_i$|X], MPI-tree($x_i$), j);
      /* $X = x_1, x_2, \ldots, x_m$ is the original melody sequence */
      /* [$x_i$|X] is an item-suffix melody sequence with item-suffix $x_i$*/
3:  end for

Subroutine MPI-tree-maintenance
Input: An item-suffix melody sequence $<x_i, x_{i+1}, \ldots, x_m>$ ($i$=1, 2, …, m), MPI-tree($x_i$) and the
       current block-id j;
Output: The modified MPI-tree($x_i$), where $i$=1, 2, ..., m;

1:  foreach item $x_i$ do /* $i$=1, 2, …, m */
2:    if MPI-tree has a child Y such that Y.item-id = $x_i$.item-id then
3:      Y.support = Y.support +1; /*increment Y's support by one*/
5:    else
6:      create a new node Y = (item-id, 1, j, node-link);
        /* initialize the Y's support to 1, and link its parent link to MPI-tree, and its node-link
        linked to the nodes with same item-id via the node-link structure. */
7:    end if
8:  end for
```

Fig. 10. Algorithm of MMS_{LMS}-summary construction.

the candidate (*cef*) is a maximal frequent item-set, since its support is 3, and it is not a sub-structure of any other maximal melody structures within the MMS_{LMS}-list. Now, MMS_{LMS}-mine stores the maximal item-set (*cef*) into the MMS_{LMS}-list.

(3) MMS_{LMS}-mine starts on the third entry *d* and generates a maximal frequent 2-item-set (*df*). We store this item-set (*df*) into the MMS_{LMS}-list because it is not a sub-structure of any other maximal melody structures within the current MMS_{LMS}-list.

```
Subroutine MMS_LMS-summary-pruning
Input: An MMS_LMS-summary, a user-specified minimum support threshold, minsup, and the
       current length of LMS, |LMS|;
Output: An MMS_LMS-summary which contains the set of all frequent melody structures.

1: foreach entry $x_i$ ($i$=1, 2, …, $d$) ∈ CMB, where $d$ =|CMB| do
2:   if $x_i$ .support < minsup |LMS| then /* $x_i$ is not a frequent item */
3:     delete those nodes (item-id = $x_i$) via node-link structure;
4:     merge the fragmented sub-trees;
       /* a simple way is to reinsert or to join the remainder sub-trees into the MPI-tree*/;
5:    delete MPI-tree($x_i$);
6:    delete the entry $x_i$ from the CMB;
7:   end if
8: end for
```

Fig. 10 (*continued*)

```
Algorithm 2 (MMS_LMS-mine)

Input: A current MMS_LMS-summary, the current length of landmark melody stream |LMS|,
       and a minimum support threshold minsup.
Output: A temporal-pattern-list, MMS_LMS-list, of maximal melody structures.

1:  MMS_LMS-list = ∅;
2:  foreach entry e in the current CMB do
3:    do generate a candidate maximal melody structure E with size |E|
         /* |E| = 1+|CMB-table(e) */
4:       counting E.support by traversing the MPI-tree(e);
5:       if E.support    minsup |LMS| then
6:         if E ∉ MMS_LMS-list and E is not a substructure of any other maximal frequent
              structures contained into the MMS_LMS-list then
7:              add E into the MMS_LMS-list;
8:              remove E's substructures from the MMS_LMS-list;
9:           end if
10:      else /* if E is not a frequent melody structure*/
11:           enumerate E into melody substructures with size |E|−1;
12:      end if
13:    until MMS_LMS-mine find the set of all maximal frequent structures with respect to the
      item e;
14: end for
```

Fig. 11. Algorithm of MMS_{LMS}-mine.

(4) On the fourth entry *e*, since its maximal melody item-set (*ef*) is a sub-structure of previous maximal melody item-set (*cef*), MMS_{LMS}-mine does not store it into the MMS_{LMS}-list.

(5) Finally, MMS_{LMS}-mine computes the entry *f*, and generates a maximal frequent 1-item-set (*f*) directly, since the CMB-table(*f*) is empty. MMS_{LMS}-mine does not store it into the MMS_{LMS}-list, because it is a sub-structure of a generated maximal item-set (*cef*).

In conclusion, the *Maximal Type I Melody Structures* determined by algorithm MMS_{LMS} are (*a*), (*cef*) and (*df*). Now, we describe the mining principle of maximal melody item-strings, i.e., *Maximal Type II Melody Structures*, as below. MMS_{LMS}-mine generates maximal melody item-strings from the current MMS_{LMS}-summary as shown in Fig. 9 by a depth-first-search (DFS) approach. Hence, the *Maximal Type II Melody Structures* determined by algorithm MMS_{LMS} are $\langle a\rangle$, $\langle c\rangle$, $\langle d\rangle$ and $\langle ef\rangle$. Note that $\langle f\rangle$ is not maximal melody item-string since it is a sub-string of the existing maximal melody 2-item-string $\langle ef\rangle$.

Based on the algorithm MMS_{LMS}-mine in Fig. 10, we have the following lemma.

Lemma 2. *A melody structure is a maximal melody structure if and only if it is generated by algorithm MMS_{LMS}-mine.*

Proof. Algorithm MMS_{LMS}-mine is composed of two major steps: *frequent melody structure selection* (step 1) and *maximal melody structure verification* (step 2). These steps are performed in sequence. First of all, in the step of frequent melody structure selection, MMS_{LMS}-mine finds frequent melody structure based on the *Apriori property* if any length i-item-set (or i-item-string) is not frequent, its length $(i+1)$-item-set (or $(i+1)$-item-string) can never be frequent. That means MMS_{LMS}-*mine does not miss any frequent melody structures*. Next, in step 2, MMS_{LMS}-mine checks the frequent melody structures generated from step 1 against the maximal melody structures of the MMS_{LMS}-list, a temporal pattern list of maximal melody structures. If this frequent melody structure is a sub-structure (i.e., *sub-set* or *sub-string*) of any other structures within the MMS_{LMS}-list, then it is not a maximal melody structure according to the Definition 5; otherwise it is a candidate maximal melody structure before the next execution of step 2. Repeating step 1 and step 2, MMS_{LMS}-mine can generate all the maximal melody structures contained in the MMS_{LMS}-list. Hence, we have the lemma: *a melody structure is a maximal melody structure if and only if it is generated by algorithm MMS_{LMS}-mine*.

Space complexity analysis: The space requirement of MMS_{LMS}-summary consists of two parts: the *working space* needed to create a CMB and the CMB-tables, and the *storage space* needed to maintain the set of MPI-trees. Assume that CMB contains k frequent chord-sets such as $e_1,e_2,\ldots,e_i,\ldots,e_k$ at any time. Based on the Theorem 1, we know that there are at most $C^k_{\lceil k/2\rceil}$ maximal frequent chord-sets in the landmark melody stream seen so far. If we construct the MMS_{LMS}-summary for all these maximal frequent melody structures, the maximum height of all the MPI-trees is $\lceil k/2\rceil$. There are $1+C^{k-1}_1+C^{k-1}_2+\cdots+C^{k-1}_{\lceil k/2\rceil-1}$ nodes in the MPI-tree(e_1), where the value 1 indicates the root node e_1 of the MPI-tree(e_1), and $C^{k-1}_1+C^{k-1}_2+\cdots+C^{k-1}_{\lceil k/2\rceil-1}$ are internal and leaf nodes of the MPI-tree(e_1). Moreover, there are $1+C^{k-2}_1+C^{k-2}_2+\cdots+C^{k-2}_{\lceil k/2\rceil-1}$ nodes in the MPI-tree(e_2), …, $1+C^{k-i}_1+C^{k-i}_2+\cdots+C^{k-i}_{\lceil k/2\rceil-1}$ nodes in the MPI-tree(e_i), $1+C^{k-(k-1)}_1$ nodes in the MPI-tree(e_{k-1}), and 1 (root) node in the MPI-tree(e_k). Thus, the total number of nodes of MPI-trees in the MMS_{LMS}-summary is

$$\begin{aligned}
&(1+C^{k-1}_1+C^{k-1}_2+\cdots+C^{k-1}_{\lceil k/2\rceil-1})\\
&\quad+(1+C^{k-2}_1+C^{k-2}_2+\cdots+C^{k-2}_{\lceil k/2\rceil-1})+\cdots\\
&\quad+(1+C^{k-i}_1+C^{k-i}_2+\cdots+C^{k-i}_{\lceil k/2\rceil-1})+\cdots\\
&\quad+(1+C^{k-(k-1)}_1)+1\\
&=(C^{k-1}_0+C^{k-1}_1+C^{k-1}_2+\cdots+C^{k-1}_{\lceil k/2\rceil-1})\\
&\quad+(C^{k-2}_0+C^{k-2}_1+C^{k-2}_2+\cdots+C^{k-2}_{\lceil k/2\rceil-1})+\cdots\\
&\quad+(C^{k-i}_0+C^{k-i}_1+C^{k-i}_2+\cdots+C^{k-i}_{\lceil k/2\rceil-1})+\cdots\\
&\quad+(C^{k-(k-1)}_0+C^{k-(k-1)}_1)+C^{k-k}_0.
\end{aligned}$$

This number equals $C^k_1+C^k_2+\cdots+C^k_{\lceil k/2\rceil}$ based on Pascal's Identity: let x and y be positive integers with $x\geqslant y$. Then $C^{x+1}_y=C^x_{y-1}+C^x_y$.

Moreover, the worst case working space requires at most $(k^2+k)/2$ entries, which is based on the process of *Sequence-Projection*. Thus, the space requirement of MMS_{LMS}-summary is $(k^2+k)/2+C^k_1+C^k_2+\cdots+C^k_{\lceil k/2\rceil}$. Finally, the upper bound of space requirement is $O(2^k)$. □

The worst case space complexity of algorithm MMS_{LMS} can be analyzed in terms of melody sequence size as described below. Assume that the average melody sequence size is m, the current length of the landmark melody stream is N, and the minimum support threshold is *minsup*. The space requirement of algorithm $\mathrm{MMS_{LMS}}$ is composed of two parts, *working space* and *storage space*. The working space is used to store the CMB and CMB-tables and the storage space is used to store the MPI-trees. The working space requirement is $m + (m - 1) + (m - 2) + \cdots + 1$ and the storage space requirement is also about $m + (m - 1) + (m - 2) + \cdots + 1$. Hence, the space requirement of $\mathrm{MMS_{LMS}}$ for inserting a melody sequence with average size m into $\mathrm{MMS_{LMS}}$-summary is $2[m + (m - 1) + (m - 2) + \cdots + 1] = m^2 + m$. Hence, the space requirement of the stream generated so far in the worst case is $N \cdot (m^2 + m)$. Note that in the analysis, we assume that the $sminsup \cdot N$ is just one and therefore every item of the incoming melody sequence is a frequent item, which is the worst case. However, we know that the value of N increases as time progresses. Hence, the pruning mechanism of $\mathrm{MMS_{LMS}}$-summary is deployed to limit the memory requirement not to exceed an upper bound.

From the space complexity analysis, it is not surprising to find that the space complexity grows exponentially into the number of frequent items in the CMB, as all frequent item-sets are represented in the data structure. It is also the solution space of the problem.

Time complexity analysis: From the construction process of $\mathrm{MMS_{LMS}}$-summary, we can see that exactly *one scan* of a landmark melody stream is required. The cost (denoted by *Time-cost*(S)) of inserting a melody sequence S into the $\mathrm{MMS_{LMS}}$-summary by sequence projection is $|S| + (|S| - 1) + \cdots + 1 = (|S|^2 + |S|)/2$; that is $O(|freq(S)|^2)$, where $freq(S)$ is the set of frequent items in the melody sequence S. Note that $|freq(S)| \leqslant |S|$, where $|S|$ denotes the size of the melody sequence S.

Because the items within the CMB are frequent items, therefore, the cost of inserting a melody sequence S can be stated in terms of the size of CMB. $Time\text{-}cost(S) = O(|S'|^2)$, where $|S'|$ is the number of chord-sets of melody sequence S within the CMB. In the worst case, if the melody sequence S contains all the frequent items within the CMB, $Time\text{-}cost(S) = O(|\mathrm{CMB}|^2)$.

4. Experimental results

In this section, we first describe the data and experiment set-up used to evaluate the performance of the proposed algorithm, and then report our experimental results.

4.1. Synthetic data and experiment set-up

To evaluate the performance of $\mathrm{MMS_{LMS}}$ algorithm, two experiments are performed. The experiments were carried out on the IBM synthetic market-basket test data generator proposed by Agrawal and Srikant (1994). Two data streams, denoted by *S10.I5.D1000K* and *S30.I15.D1000K*, of size 1 million melody sequences each are studied. The first one, *S10.I5.D1000K* with 1 *K* unique items, has an average melody sequence size of 10 with average maximal potentially frequent structure size of 5. The second one, *S30.I15.D1000K* with 10 *K* unique items, has an average melody sequence size of 30 with average maximal potentially frequent structure size of 15. In all experiments, the melody sequences of each datasets are looked up in sequence to simulate the environment of a landmark melody stream. All the experiments are performed on a 1066-MHz Pentium III processor with 128 megabytes main memory, running on Microsoft Windows XP. In addition, all the programs are written in Microsoft/Visual C++ 6.0.

4.2. Experimental results

In the first experiment, two primary factors, *memory* and *execution time*, are examined in the online mining of a landmark melody stream, since both should be bounded online as time advances. In Fig. 12(a), the execution time grows smoothly as the dataset size increases. This is because the average execution time of dataset $S10.I5$ and $S30.I15$ are about 12 and 25 s per block respectively, where a block is composed of 50,000 melody sequences. In other words, the computation time of dataset $S10.I5$ by algorithm $\mathrm{MMS_{LMS}}$ is 12 s every 50,000 melody sequences, and for dataset $S30.I15$ is 25 s every 50,000 melody sequences. Hence, it grows smoothly as the dataset size increases. The memory usage in Fig. 12(b) for both

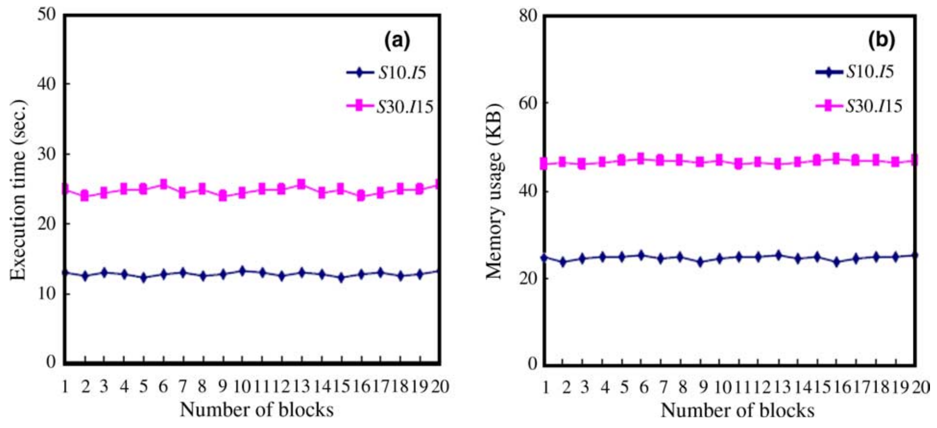

Fig. 12. Required resources for synthetic datasets: (a) execution time and (b) memory.

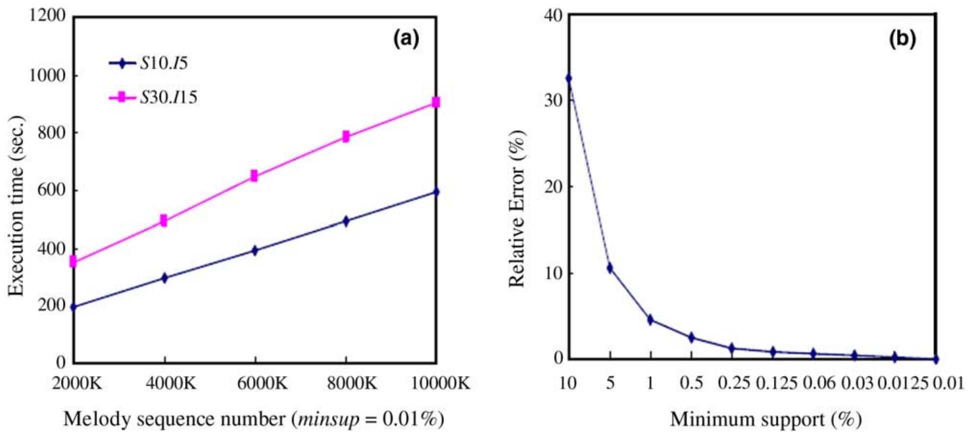

Fig. 13. (a) Linear scalability of the data stream size and (b) relative error of mining results.

synthetic datasets is stable as time progresses, indicating the feasibility of algorithm $\mathrm{MMS_{LMS}}$. Note that the synthetic landmark melody stream is partitioned into blocks with size 50 K.

In the second experiment, we investigate the *scalability* and *relative error* of algorithm $\mathrm{MMS_{LMS}}$ with respect to varying minimum supports. The relative error is defined as the difference between the measured support and the actual support estimation divided by the actual support. In Fig. 13(a), the execution time grows *smoothly* as the dataset increases (assume *minsup* = 0.01%) indicating linear scalability. Fig. 13(b) shows that the relative error decreases as *minisup* decreases, i.e., as the *size* of CMB *decreases*. Generally, the more frequent items are maintained in the CMB, the more accurate the mining result is.

5. Conclusions

In this paper, we proposed a single-pass algorithm, $\mathrm{MMS_{LMS}}$, to discover and maintain all maximal melody structures in a landmark model that contains all the melody sequences in a data stream. In the $\mathrm{MMS_{LMS}}$ algorithm, an efficient in-memory summary data structure, $\mathrm{MMS_{LMS}}$-summary, is developed to record all maximal frequent structures in the current landmark model. In addition, $\mathrm{MMS_{LMS}}$ uses a space-efficient scheme, the Chord-set Memory Border (CMB), to guarantee the upper-bound of space requirements of mining maximal melody sequences in a streaming environment. Theoretical analysis and experimental results with synthetic data show that MMS_{LMS} algorithm can meet the performance requirements of data stream mining: one-scan, bounded-space and real time. Further work includes online mining maximal melody structures in count-based and time-based sliding window that contains the most recent melody sequences in a data stream.

Acknowledgements

The authors thank the reviewers' precious comments for improving the quality of the paper. The research is supported by National Science Council of R.O.C. under grant no. NSC93-2213-E-009-043.

References

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules. In: Proceedings of 20th International Conference on Very Large Data Bases, pp. 487–499.

Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J., 2002. Models and issues in data stream systems. In: Proceedings of 21th ACM Symposium on Principles of Database Systems, pp. 1–16.

Bakhmutora, V., Gusev, V.U., Titkova, T.N., 1997. The search for adaptations in song melodies. Computer Music Journal 21 (1), 58–67.

Fisher, M.J., Salzberg, S.L., 1982. Finding a majority among n votes: solution to problem 81-5. Journal of Algorithms 3 (4), 362–380.

Hsu, J.L., Liu, C.C., Chen, A.L.P., 2001. Discovering non-trivial repeating patterns in music data. IEEE Transactions on Multimedia 3 (3), 311–325.

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

Karp, R.M., Papadimitrious, C.H., Shanker, S., 2003. A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems 28 (1), 51–55.

Shan, M.-K., Kuo, F.-F., 2003. Music style mining and classification by melody. IEICE Transactions on Information and Systems E86-D (4), 655–659.

Yoshitaka, A., Ichikawa, T., 1999. A survey on content-based retrieval for multimedia databases. IEEE Transactions on Knowledge and Data Engineering 11 (1), 81–93.

Zhu, Y., Kankanhalli, M.S., Xu, C., 2001. Pitch tracking and melody slope matching for song retrieval. In: Proceedings of the Second IEEE Pacific Rim Conference on Multimedia: Advances in Multimedia Information, pp. 530–537.