An Efficient Approach for Mining Top-K Fault-Tolerant Repeating Patterns*

Jia-Ling Koh and Yu-Ting Kung

Department of Information and Computer Education,
National Taiwan Normal University, Taipei, Taiwan 106, R.O.C
jlkoh@ice.ntnu.edu.tw

Abstract. In this paper, an efficient strategy for mining top-K non-trivial fault-tolerant repeating patterns (FT-RPs in short) with lengths no less than *min_len* from data sequences is provided. By extending the idea of appearing bit sequences, fault-tolerant appearing bit sequences are defined to represent the locations where candidate patterns appear in a data sequence with insertion/deletion errors being allowed. Two algorithms, named **TFTRP-Mine**(**T**op-K non-trivial **FT-RP**s **Mining**) and **RE-TFTRP-Mine** (**RE**finement of **TFTRP-Mine**), respectively, are proposed. Both of these two algorithms use the recursive formulas to obtain the fault-tolerant appearing bit sequence of a pattern systematically and then the fault-tolerant frequency of each candidate pattern could be counted quickly. Besides, RE-TFTRP-Mine adopts two additional strategies for pruning the searching space in order to improve the mining efficiency. The experimental results show that RE-TFTRP-Mine outperforms TFTRP-Mine algorithm when *K* and *min_len* are small. In addition, more important and implicit repeating patterns could be found from real music objects by adopting fault tolerant mining.

1 Introduction

Repeating patterns represent the important sub-patterns in a data sequence because they appear repeatedly. There have been many approaches proposed for mining repeating patterns[1][3][4]. However, in most approaches, only exact pattern matching was considered during the mining process. It may cause some implicit repeating patterns not being found because of insertion/deletion errors occurring. For example, suppose two data sequences: S1="ACDE…ACEDE…", and S2 ="ACD E… ADE…" are given. The pattern "ACEDE" approximately matches "ACDE" with one insertion error in S1. Besides, the pattern "ADE" approximately matches "ACDE" with one deletion error in S2. However, the exact matching approach will lost the implicit repeating pattern "ACDE" in these two sequences.

To solve the above problem, this paper focuses on the strategy for mining "*fault-tolerant*" repeating patterns, *FT-RP*s in short. In other words, the insertion/deletion errors are allowed when counting the appearing frequency of a pattern. Besides, to avoid duplicated information and many short patterns being found, only "non-trivial" FT-RPs, i.e., those FT-RPs containing no super-pattern with the same fault-tolerant

* This work was partially supported by the R.O.C. N.S.C. under Contract No. 94-2213-E-003-010.

M.L. Lee, K.L. Tan, and V. Wuwongse (Eds.): DASFAA 2006, LNCS 3882, pp. 95 – 110, 2006.

frequency, and their lengths no less than a given *min_len* are mined out. Moreover, by giving the desired number of non-trivial FT-RPs to be mined, we propose an approach of mining "top-K non-trivial fault-tolerant repeating patterns with length no less than min_len" to avoid finding a huge amount of non-representative patterns.

A data structure called *correlative matrix* was proposed in [3] to aid the process for extracting repeating patterns in a music object. The main disadvantage of this approach is that the processing cost is proportional to the square of the length of the music object. To solve this problem, the same authors developed the *String-Join* algorithm [1] to extract the *non-trivial* repeating patterns in a music object. In [4], the representation of *bit index sequence* was designed to characterize note sequences of music objects. In the mining process, the frequency of a candidate pattern was obtained by performing **shift** and **and** operations on bit sequences and then counting the number of 1s in the resultant bit sequence. Therefore, frequency checking could be performed quickly.

Fault-tolerant data mining would discover more general and useful information for real-world dirty data. The problem of fault-tolerant frequent patterns (itemsets) was defined and solved in [6] by proposing FT-Apriori algorithm. Similar to the Apriori-like algorithms, FT-Apriori algorithm suffered from generating a large number of candidates and scanning database repeatedly. This problem became worse when the fault tolerance was increasing or the support thresholds were decreasing. To speed up the mining of fault tolerant frequent patterns, we proposed an algorithm named FFT-Mine (Fast Fault Tolerant frequent patterns Mining) in [5]. By extending the form of appearing vectors, the fault-tolerant appearing vectors were defined to represent the distribution that the candidate patterns were contained in database with fault tolerance. FFT-Mine algorithm provided a systematically method to reduce the number of operations performed on bit vectors to get the fault-tolerant appearing vectors of candidates. Then whether a candidate is a fault-tolerant frequent itemset could be judged quickly.

When mining frequent patterns, it is difficult for users to set an appropriate minimum support threshold without knowing the distribution of data in the database. Moreover, if long patterns exist in a database, the mining result may return many short or tedious patterns with duplicated information. To prevent the above problems occurring, [2] proposed a TFP algorithm to discover top-K frequent closed patterns with length no less than *min_l*. For solving the similar problems when mining frequent sequential patterns, TSP algorithm was proposed in [7]. It adopted the similar idea proposed in TFP algorithm to raise the minimum support during the mining process for discovering top-K closed sequential patterns. Then the searching space would be pruned dramatically to speed up the mining process.

In summarizing the interesting strategies proposed in the related works, an efficient way of mining top-K non-trivial fault-tolerant repeating patterns (FT-RPs in short) with length no less than *min_len* for data sequences is proposed in this paper. By extending the idea of appearing bit sequences, fault-tolerant appearing bit sequences are defined to represent the locations where candidate patterns appear in a data sequence with insertion/deletion errors allowed. Then the fault-tolerant frequency of a candidate pattern could be counted from its fault-tolerant appearing bit sequence quickly. The recursive formulas are designed for obtaining the fault-tolerant appearing bit sequence of a pattern systematically in order to eliminate the duplicate computations. Two algorithms, named **TFTRP-Mine** and **RE-TFTRP-Mine**, respectively, are proposed. The TFTRP-Mine algorithm generates candidate patterns by performing a depth-first searching approach. The RE-TFTRP-Mine algorithm adopts two additional strategies to increase the mining efficiency. The first one is to assign priorities of found repeating patterns for generating candidates according to their fault-tolerant frequencies. Moreover, the minimum frequency is raised dynamically when K numbers of FT-RPs have been found. The experimental results show these two strategies will prune the searching space dramatically when K is small proportional to the number of whole FT-RPs.

This paper is organized as follows. Section 2 defines the relative terms used in this paper. The appearing bit sequences and the way of getting fault-tolerant appearing bit sequences are introduced in Section 3. Section 4 describes the whole processing steps of TFTRP-Mine and RE-TFTRP-Mine algorithms. The performance evaluation of proposed algorithms is shown in Section 5. Finally, in Section 6, we propose the conclusion and feature works of this paper.

2 Preliminaries

[*Def.* 2.1] Let E={l_1, l_2, …, l_k} denote the set of data items in a specific application domain. $DSeq$=$D_1D_2…D_n$ is a **data sequence**, where $D_i \in$ E(i=1…n) denoting the data item on the ith position of the sequence. The length of $DSeq$ is denoted as $|DSeq|$.

[*Def.* 2.2] Let S_1 and S_2 denote two data sequences, where $S_1=X_1X_{2...}X_m$ and $S_2=Y_1Y_{2...}Y_n$. S_2 is a **sub-sequence** of S_1 iff there exists an integer sequence i_1, i_2, ..., i_n such that $1\leq i_1 \leq i_{2\,...} \leq m$ and $X_{ik} = Y_k$ for k = 1to n.

[*Def.* 2.3] Given a data sequence $DSeq$=$D_1D_2…D_n$ and another data sequence (also named a pattern) $P=P_1P_{2...}P_m$, P **appears** in $DSeq$ on position i iff there exists an integer $1\leq i \leq$ n, such that $D_iD_{i+1}…D_{i+m-1}= P_1P_{2...}P_m$. It is also said $DSeq$ **contains** P on position i and P is a **sub-pattern** of $DSeq$. The **frequency** of a pattern P in $DSeq$ is the number of various positions in $DSeq$ where $DSeq$ contains P.

[*Def.* 2.4] A data sequence $DSeq$=$D_1D_2…D_n$ is said to **FT-contain** pattern $P=P_1P_{2...}P_m$ (m≥2) on position i **with δ insertion errors** iff there exist an integer $1\leq i \leq$ n, such that $D_i=P_1$, $D_{(i+m-1)+\delta}=P_m$, and P is a sub-sequence of $D_iD_{i+1}…D_{(i+m-1)+\delta}$. Given a fault tolerance $\boldsymbol{\delta_I}$ ($\boldsymbol{\delta_I}$ >0), $DSeq$ is said to **insertion FT-contain** pattern P under fault tolerance $\boldsymbol{\delta_I}$, denoted as **IFT-contain** in short, iff $DSeq$ FT-contains P with **δ** insertion errors and $\boldsymbol{\delta \leq \delta_I}$. In other words, there exists a sub-pattern of $DSeq$ starting from position i which is gotten by inserting at most $\boldsymbol{\delta_I}$ data items in the *middle* of P. The pattern is also said to **IFT-appear** in $DSeq$.

[Example 2.1] Suppose $DSeq$=ABCDABCA, and $\boldsymbol{\delta_I}$**=2**. Given patterns P_1=ABCA, P_2=BCAC, and P_3=ABBC. According to [Def. 2.4], $DSeq$ FT-contains P_1 on position 1 with 1 insertion error. Besides, $DSeq$ FT-contains P_1 on position 5 with 0 insertion error. Similarly, $DSeq$ FT-contains P_2 on position 2 with two insertion errors. Therefore, $DSeq$ IFT-contains P_1 and P_2. However, P_3 doesn't IFT-appear in $DSeq$.

[*Def.* 2.5] A data sequence $DSeq$=$D_1D_2…D_n$ is said to **FT-contain** a pattern $P=P_1P_{2...}P_m$ (m>δ) on position i **with δ deletion errors** iff there exist an integer

$1 \leq i \leq$ n, such that $D_i D_{i+1} \ldots D_{(i+m-1)-\delta}$ is a sub-sequence of P. Given a fault tolerance $\boldsymbol{\delta_D}(\boldsymbol{\delta_D} > \mathbf{0})$, *DSeq* is said to **deletion FT-contain** pattern P, denoted as **DFT-contain** in short, iff *DSeq* FT-contains P on position i with $\boldsymbol{\delta}$ deletion errors, where $D_i = P_1$ and $\boldsymbol{\delta} \leq \boldsymbol{\delta_D}$. That is, there exists a sub-pattern of *DSeq* starting from position i which is gotten by deleting *at most* $\boldsymbol{\delta_D}$ data items from P except the first data item. The pattern P is also said to **DFT-appear** in *DSeq*.

[Example 2.2] Suppose *DSeq*=ABCBCA, and $\boldsymbol{\delta_D} = \mathbf{3}$. Given patterns P_1=BCDA and P_2=EFB. *DSeq* FT-contains P_1 on position 4 with 1 deletion error (by deleting "D" from P_1). Therefore, P_1 DFT-appears in *DSeq*. Although *DSeq* FT-contains P_2 on positions 2 and 4, respectively, with 2 deletion errors, P_2 doesn't DFT-appear in *DSeq* because the deletion error on the first data item of P_2 is not allowed.

[*Def.* 2.6] The **fault tolerant frequency** of a pattern P in *DSeq*, denoted as $FT\text{-}freq_{DSeq}(P)$, is the number of various positions in *DSeq* where *DSeq* IFT/DFT-contains P. The pattern P is named a **fault-tolerant repeating pattern, FT-RP** in short, if and only if $FT\text{-}freq_{DSeq}(P) \geq$ a required minimum frequency *min_freq*.

[*Def.* 2.7] A FT-RP P is a **non-trivial FT-RP** if there does not exist any FT-RP P' such that P is a sub-pattern of P', and $FT\text{-}freq_{DSeq}(P') = FT\text{-}freq_{DSeq}(P)$.

3 Bit Sequence Representation

In this section, section 3.1 will introduce the design of appearing bit sequences. How to apply the *appearing bit sequences* of patterns to compute the frequency of candidate patterns with fault tolerance quickly is introduced in section 3.2 and 3.3.

3.1 Appearing Bit Sequences

For each kind of data item N in the data sequence, N has a corresponding *appearing bit sequence* (denoted as $Appear_N$). The length of each appearing bit sequence equals the length of the data sequence. The leftmost bit is numbered as bit 1 and the numbering increases to the rightmost bit. If some data item appears on the ith position of the data sequence, bit i in the appearance bit sequence of this data item is set to be 1; otherwise, it is set to be 0. A bit index table is used to store the appearing bit sequences for all the data items in the data sequence. Therefore, the frequency of a data item is obtained according to the number of bits with value 1 in its appearing bit sequence, without needing to scan the data sequence repeatedly. The idea is also applicable for a longer pattern as explained in the following example.

[Example 3.1] The bit index table of "ABCDABCACDEEABCCDEAC" is given as shown in Table 1.

1) Suppose we would like to get $Appear_{AB}$. A position i where "AB" appears implies "A" must appear on position i and "B" appears on the next position (i+1).
   Step1. Get $Appear_B$=01000100000001000000 from Table 1.

Table 1. The bit index table of *DSeq*

Data Item	Appearing Bit Sequence
A	10001001000010000010
B	01000100000001000000
C	00100010100000110001
D	00010000010000001000
E	00000000001100000100

Step2. Perform **left shift** 1 (=|AB|-1) bit operation on $Appear_B$ (shift bit (i+1) to bit i, where 1≤i≤19, and set bit 20 to be 0), $L\_shift(Appear_B, 1)$ = 10001000000010000000.

Step3. $Appear_{AB} = Appear_A \wedge L\_shift(Appear_B, 1)$ = 10001000000010000000.

2) Suppose we would like to get $Appear_{ABC}$ after getting $Appear_{AB}$. A position i where "ABC" appears implies "AB" must appears on position i and "C" appears on position i+2.

Step1. Obtain $Appear_C$=00100010100000110001 from Table 1.

Step2. Perform **left shift** 2 (=|ABC|-1) bits on $Appear_C$, $L\_shift(Appear_C, 2)$ = 10001010000011000100.

Step3. $Appear_{ABC} = Appear_{AB} \wedge L\_shift(Appear_C, 2)$ = 10001000000010000000.

Accordingly, the frequency of "ABC" in *DSeq* equals the number of bits with value 1 in $Appear_{ABC}$ (that is 3 in this case).

Suppose pattern $P=P_1P_{2...}P_m$ (m≥2), where P_i (i=1, …,m) is a data item. Let $P'=P_1P_{2...}P_{m-1}$ and $X=P_m$. Then $Appear_P$ could be deduced from $Appear_{P'}$ and $Appear_X$ according to the recursive formula 3.1 shown below.

If |P|=1, $Appear_P = Appear_P$;

Otherwise, $Appear_P = Appear_{P'} \wedge L\_shift(Appear_X, |P|-1)$. (3.1)

The function $L\_shift(b, n, c)$ performs **left shift** n bits on b, and the rightmost bits on b are filled with constant c(c=0 or 1). If the parameter c is omitted from the function, the default value of c is set to be 0.

3.2 Appearing Bit Sequences with Insertion Fault Tolerance

By extending appearing bit sequences, the fault-tolerant appearing bit sequences are designed to represent the appearing positions of a pattern with fault tolerance. Given a fault-tolerance $\delta(\delta_I$ or $\delta_D)$, the fault-tolerant appearing bit sequence of a pattern P in a data sequence, denoted as $FT\text{-}Appear_P^+(\delta)/FT\text{-}Appear_P^-(\delta)$, represents the positions where the data sequence IFT/DFT-contains P.

By considering the insertion fault tolerance, the appearing bit sequence of a pattern P with E numbers of insertion errors, denoted as $Appear_P^+(E)$, is defined. The bits with value 1 in $Appear_P^+(\delta)$ represent those positions where the data sequence FT-contains P with E insertion errors. According to [***Def.* 2.4**], there are (δ_I+1) situations that a pattern P IFT-appears in *DSeq* under fault tolerance δ_I. That is, *DSeq* FT-contains P with 0, 1, 2, …, or δ_I insertion errors. In other words, performing δ_I **or** operations on (δ_I+1) appearing bit sequences: $Appear_P^+(0)$, $Appear_P^+(1)$, $Appear_P^+(2)$,

…, and $Appear_P^+(\delta_I)$, $FT\text{-}Appear_P^+(\delta_I)$ could be obtained. According to the definition, $Appear_P^+(E)$ with |P|=1 is obtained from the following rule:

[Rule 3.1] Suppose the insertion fault-tolerance is set to be δ_I. If $|P|=1$, $Appear_P^+(E)=0$ for all $1 \le E \le \delta_I$. (3.2)

The remaining problem is how to get $Appear_P^+(E)$ for |P|>1 and $0 \le E \le \delta_I$. Since $Appear_P^+(0)$ represents the locations where *DSeq* FT-contains *P* with *zero* insertion error, the way of getting $Appear_P^+(0)$ is the same as getting $Appear_P$. When $1 \le E \le \delta_I$, $Appear_P^+(E)$ could be obtained by performing bit operations on appearing bit sequences of the prefix of *P* with length |P-1| and the last data item in P according to the following lemma.

[Lemma 3.1] Given a pattern $P=P_1P_{2\ldots}P_m$, where P_i (i=1,…,m) is a data item. Let P' denote $P_1P_{2\ldots}P_{m-1}$ and *X* denote P_m. *DSeq* FT-contains pattern *P* with *E* insertion errors on position *i*, iff *DSeq* FT-contains pattern P' with *k* insertion errors on position *i* $(0 \le k \le E)$ and *X* appears on position i+(|P|+E)-1.

Proof. P' appears in *DSeq* from position *i* to (i+|P|-1)+k (with k insertion errors) and E-k insertion errors occurs between P' and *X*. Besides, |P'|+1=|P|. It induces that *X* appears on position (i+|P|-1)+k+(E-k)+1=i+(|P'|+1)+E-1= i+(|P|+E)-1.

In other words, *X* must appear on the (|P|+E-1)*th* position on the right hand side of position *i*. Therefore, the way of getting $Appear_P^+(E)$ is expressed as the following recursive formula for $0<E\le\delta_I$.

If |P|=1, $Appear_P^+(E)=0$;

Otherwise, $Appear_P^+(E)=(\bigvee_{k=0}^{E} Appear_{P'}^+(k)) \wedge L\_shift(Appear_X, |P|+E-1)$. (3.3)

To combine Formulas (3.1) and (3.3), a recursive function of getting $Appear_P^+(E)$, where $0<E\le\delta_I$ is defined as follows.

[*Def.* 3.1] Recursive function of getting $Appear_P^+(E)$**:** Suppose a pattern $P=P_1P_{2\ldots}P_m$ is given, where P_i (i=1,…,m) is a data item. Let P' denote $P_1P_{2\ldots}P_{m-1}$ and *X* denote P_m. When insertion fault tolerance δ_I is given, $Appear_P^+(E)$ is obtained from the following recursive function for $0\le E\le\delta_I$.

If |P|=1, **then** $Appear_P^+(0)= Appear_P$; $\forall 1\le E\le\delta_I$, $Appear_P^+(E)=0$;

Else $Appear_P^+(E)= (\bigvee_{k=0}^{E} Appear_{P'}^+(k)) \wedge L\_shift(Appear_X, |P|+E-1)$.

[Example 3.2] Suppose δ_I is set to be 1. According to the bit index table shown in Table 1, the process of getting $Appear_{AB}^+(1)$ and $Appear_{ABC}^+(1)$ is shown as follows.

(1) $Appear_{AB}^+(1)$

Step1. Get $Appear_B$ = 010001000000001000000 from the bit index table.

Step2. Perform an **or** operation on $Appear_A^+(0)$ and $Appear_A^+(1)$. According to formula (3.2), $Appear_A^+(1)=0$, and $Appear_A^+(0)= Appear_A$.

$s = Appear_A^+(0) \vee Appear_A^+(1)$=100010010000100000010.

Step3. Perform **left shift** 2 (= |AB|+1-1) bits on $Appear_B$,

$t = L\_shift(Appear_B, 2) =$ 00010000000100000000.

Step4. Perform an **and** operation on s and t to get $Appear_{AB}^+(1)$. Thus the resultant bit sequence: $s \wedge t$= 00000000000000000000.

(2) $Appear_{ABC}^+(1)$

Step1. Get $Appear_C$ = 00100010100000110001.

Step2. Perform an **or** operation on $Appear_{AB}^+(0)$ and $Appear_{AB}^+(1)$. Since $Appear_{AB}^+(0)$ is gotten based on formula (3.1) and $Appear_{AB}^+(1)$ is known from the previous result of this example, the resultant appearing bit sequence: $s=Appear_{AB}^+(0) \vee Appear_{AB}^+(1)$= 10001000000010000000.

Step3. Perform **left shift** 3 (=|ABC|+1-1) bits on $Appear_C$,

$t = L\_shift(Appear_C, 3) =$ 00010100000110001000.

Step4. Perform an **and** operation on s and t to get $Appear_{ABC}^+(1)$. Thus the resultant bit sequence: $s \wedge t$= 00000000000010000000.

Finally, $FT\text{-}Appear_P^+(\delta_I)$ is obtained by performing $\bigvee_{i=0}^{\delta_I} Appear_P^+(i)$. $FT\text{-}freq_{DSeq}(P)$ equals to the number of bits with value 1 in $FT\text{-}Appear_P^+(\delta_I)$. Therefore, the insertion fault-tolerant frequency of a pattern P could be counted quickly.

[Example 3.3] Follows the results shown in Example 3.1 and Example 3.2, $FT\text{-}Appear_{ABC}^+(1) = Appear_{ABC}^+(0) \vee Appear_{ABC}^+(1)$ = 10001000000010000000 and $FT\text{-}freq_{DSeq}$("ABC") = 3.

To avoid duplicate computations of **or** and **left shift** operations to get $Appear_P^+(E)$ for various E, the function of getting $Appear_P^+(E)$ is re-defined to use recurrent relations between temporary results for getting $Appear_P^+(E)$ and $Appear_P^+(E\text{-}1)$.

[*Def.* 3.2] Modified recursive function of getting $Appear_P^+(E)$**:** Suppose a pattern $P=P_1P_2...P_m$ is given. Let $P' =P_1P_2...P_{m-1}$ and X denote P_m. $Appear_P^+(E)$ is obtained from the following recursive function for $0 \leq E \leq \delta_I$.

If |P|=1, **then** $Appear_P^+(0)= Appear_P$; $\forall 1 \leq E \leq \delta_I$, $Appear_P^+(E)=0$;
Else If E =0, **then** $temp_1(E) = Appear_{P'}^+(0)$; $temp_2(E) = L\_shift(Appear_X, |P|\text{-}1)$;
Else $temp_1(E) = temp_1(E\text{-}1) \vee Appear_{P'}^+(E)$; $temp_2(E) = L\_shift(temp_2(E\text{-}1), 1)$;
$Appear_P^+(E)= temp_1(E) \wedge temp_2(E)$.

3.3 Appearing Bit Sequences with Deletion Fault Tolerance

The appearing bit sequence of a pattern P with E numbers of deletion errors is denoted as $Appear_P^-(E)$. The bits with value 1 in $Appear_P^-(E)$ represent those positions where the data sequence FT-contains P with E deletion errors.

Suppose a pattern $P=P_1P_2...P_m$ is given. Let Y denote the first data item P_1 and P" denote $P_2P_3...P_m$. $FT\text{-}Appear_P^-(\delta_I)$ represents the positions where Y appears and $DSeq$ FT-contains P" on the next positions with at most δ_D deletion errors. Therefore, when finding a position j where $DSeq$ FT-contains P" with 0, 1, 2, …, or δ_D deletion errors, if implies $DSeq$ DFT-contains P on position (j-1) if position (j-1) contains Y. In other words, after performing **or** operations on (δ_D+1) appearing bit sequences:

$Appear_{P''}^{+}(0)$, $Appear_{P''}^{+}(1)$,…, $Appear_{P''}^{+}(\delta_D-1)$, and $Appear_{P''}^{+}(\delta_D)$, then performing a **left shift** operation on the previous result, and finally performing an **and** operation with $Appear_Y$, $FT\text{-}Appear_{P}^{-}(\delta_D)$ could be obtained. Note that if $|P| \leq \delta_D+1$, when performing the left shift operation, the rightmost bit is filled with 1 because the bit is considered as "don't care" bit on the next performed **and** operation. Otherwise, 0 is filled to the rightmost bit. According to the definition, $Appear_{P''}^{-}(E)$ is obtained from the following rule for all $|P''| \leq E \leq \delta_D$:

[Rule 3.2] Suppose the deletion fault-tolerance is set to be δ_D. If $|P''| \leq \delta_D$, $Appear_{P''}^{-}(E)=0$ for all $|P''|<E\leq\delta_D$; $Appear_{P''}^{-}(E)=\text{complement}(Appear_{P''})$ for $E=|P''|$. (3.4)

Accordingly, the remaining problem is to get $Appear_{P''}^{-}(E)$ for $0 \leq E < |P''|$. Since $Appear_{P}^{-}(0)$ represents the positions where *DSeq* FT-contains P" with *zero* deletion error, it implies the same information represented in $Appear_{P''}$. Therefore, the way of getting $Appear_{P}^{-}(0)$ is the same as getting $Appear_{P''}$. When $1 \leq E \leq |P''|$, $Appear_{P''}^{-}(E)$ is obtained by performing bit operations on appearing bit sequences of the prefix of P" with length |P"-1| and the last data item in P" according to the following lemma.

[Lemma 3.2] Given a pattern $P''=P_2P_3\ldots P_m$, where ($i=2$, …,m) is a data item. Let Q denote $P_2P_3\ldots P_{m-1}$, and X denote the last data item P_m. *DSeq* FT-contains pattern P"with E deletion errors on position i, iff

1) *DSeq* FT-contains pattern Q with E deletion errors on position i and X appears on position $i+(|P''|-1-E)$, or
2) *DSeq* FT-contains pattern Q with (E-1) deletion errors on position i and FT-contains X on position $i+(|P''|-E)$ with 1 deletion error.

Proof

1) Q appears in *DSeq* from position i to $i+(|Q|-E)-1$ (with E deletion errors). If *DSeq* FT-contains P" on position i with E deletion errors, X must appear on position $i+(|P''|-1-E)$ (because $|Q|=|P''|-1$).
2) Q appears in *DSeq* from position i to $i+(|Q|-(E-1))-1= i+(|Q|-E)$ (with E-1 deletion errors). Then X is forced to be absent on position $i+(|Q|-E)+1$. That is, *DSeq* FT-contains X with 1 deletion error on position $i+(|P''|-1-E)+1=i+(|P''|-E)$.

Therefore, the way of getting $Appear_{P''}^{-}(E)$ is expressed as the following recursive function for $0 < E \leq \delta_D$.

If $|P''| < E$, $Appear_{P''}^{-}(E) = 0$;
Else if $|P''| = E$, $Appear_{P''}^{-}(E) = \text{complement}(Appear_{P''})$;
Else $Appear_{P''}^{-}(E)=(Appear_{Q}^{-}(E) \wedge \text{L\_shift}(Appear_{X}, |P''|\text{-}E\text{-}1,0))\vee$
$(Appear_{Q}^{-}(E\text{-}1) \wedge \text{L\_shift}(Appear_{X}^{-}(1), |P''|\text{-}E,1))$. (3.5)

To combine Formulas (3.1) and (3.5), a recursive function of getting $Appear_{P''}^{-}(E)$, where $0 \leq E \leq \delta_D$ is defined as follows.

[*Def.* 3.3] (Recursive function of getting $Appear_{P''}^{-}(E)$**):** Suppose a pattern $P''= P_2P_3\ldots P_m$ is given. Let Q denote $P_2P_3\ldots P_{m-1}$ and X denote P_m. When deletion fault tolerance δ_D is given, $Appear_{P''}^{-}(E)$ is obtained from the following recursive function for $0 \leq E \leq \delta_D$.

IF $|P''|=1$, **then** $Appear_{P''}^{-}(0) = Appear_{P}(E)$; $Appear_{P''}^{-}(1) = \text{complement}(Appear_{P})$;
Else if $E = 0$, **then** $Appear_{P''}^{-}(0) = Appear_{Q}^{-}(0) \wedge L\_shift(Appear_{x}, |P''|-1)$;
Else if $E > |P''|$, **then** $Appear_{P''}^{-}(E) = 0$;
Else if $E = |P''|$, **then** $Appear_{P''}^{-}(E) = \text{complement}(Appear_{P''}^{-}(0))$;
Else $Appear_{P''}^{-}(E) = (Appear_{Q}^{-}(E) \wedge L\_shift(Appear_{x}, |P''|-E-1,0)) \vee$
$(Appear_{Q}^{-}(E-1) \wedge L\_shift(Appear_{x}^{-}(1), |P''|-E,1))$.

[Example 3.4] Suppose δ_D is set to be 1. According to the bit index table shown in Table 1, the process of getting $Appear_B^{-}(1)$, $Appear_{BC}^{-}(1)$ and $Appear_{BCD}^{-}(1)$ is described as follows.

(1) $Appear_B^{-}(1) = \text{complement}(Appear_B)$.
Step1. Get $Appear_B$ = 01000100000001000000.
Step2. $Appear_B^{-}(1) = \neg Appear_B$ = 10111011111110111111.
(2) $Appear_{BC}^{-}(1) = (Appear_B^{-}(1) \wedge L\_shift(Appear_C,0,0) \vee$
$(Appear_B^{-}(0) \wedge L\_shift(Appear_C^{-}(1),1,1)$
Step1. Get $Appear_C$ = 00100010100000110001.
Step2. Perform **left shift** 0 (=|BC|-1-1) bit on $Appear_C$,
$s = L\_shift(Appear_C,0,0)$=00100010100000110001.
Step3. Perform an **and** operation on s and $Appear_B^{-}(1)$, where the result of $Appear_B^{-}(1)$ has been obtained previously. $u = s \wedge Appear_B^{-}(1)$=00100010100000110001.
Step4. $Appear_C^{-}(1) = \neg Appear_C$ =11011101011111001110.
Step5. Perform **left shift** 1 (=|BC|-1) bit on $Appear_C^{-}(1)$ (the rightmost bit is filled with 1). $t = L\_shift(Appear_C^{-}(1),1,1)$ = 10111010111110011101.
Step6. Perform an **and** operation on t and $Appear_B^{-}(0)$.
$v = t \wedge Appear_B$=00000000000000000000.
Step7. Perform an **or** operation on u and v. Then the resultant bit sequence is $w = u \vee v$ = 00100010100000110001.
(3) $Appear_{BCD}^{-}(1) = (Appear_{BC}^{-}(1) \wedge L\_shift(Appear_D,1,0) \vee$
$(Appear_{BC}^{-}(0) \wedge L\_shift(Appear_D^{-}(1),2,1)$
Step1. Get $Appear_D$ = 00010000010000001000.
Step2. Perform **left shift** 1 (=|BCD|-1-1) bit on $Appear_D$.
$s = L\_shift(Appear_D(1),1,0)$ =00100000100000010000.
Step3. Perform an **and** operation on s and $Appear_{BC}^{-}(1)$, where the result of $Appear_{BC}^{-}(1)$ has been obtained previously.
$u = s \wedge Appear_{BC}^{-}(1)$=00100000100000010000
Step4. $Appear_D^{-}(1) = \neg Appear_D$=11101111101111110111.
Step5. Perform **left shift** 2 (=|BCD|-1) bits on $Appear_D^{-}(1)$.
$t = L\_shift(Appear_D^{-}(1),2,1)$=10111110111111011111.
Step6. Perform an **and** operation on t and $Appear_{BC}^{-}(0)$.
$v = t \wedge Appear_{BC}^{-}(0)$=00000100000001000000.
Step7. Perform an **or** operation on u and v. Then the resultant bit sequence is $w = u \wedge v$ = 00100100100001010000.

Let $temp(E)$ denote the results of $\bigvee_{k=0}^{E} Appear_{P''}^{-}(k)$. To combine formulas 3.4 and 3.5, a recursive function of getting $FT\_Appear_P^{-}(\delta_D)$ is defined as follows.

[*Def.* 3.4] (Recursive function of getting $FT\_Appear_P^-(\delta_D)$**):** Suppose a pattern $P=P_1P_2\ldots P_m$ is given, where P_i (i=1,…,m) is a data item. Let Y denote P_1, P'' denote $P_2P_3\ldots P_m$, Q denote $P_2P_3\ldots P_{m-1}$, and X denote P_m. When deletion fault tolerance δ_D is given, $FT\_Appear_P^-(\delta_D)$ is obtained from the following recursive function.

If $|P| \leq \delta_D + 1$, **then** $FT\_Appear_P^-(\delta_D) = Appear_Y$**;**
Else $temp_{P'}(\delta_D - 1) = Appear_Q \vee (Appear_Q^-(\delta_D - 1) \wedge \text{L\_shift}(Appear_X, |P''| - \delta_D, 0))$;
$temp_{P'}(\delta_D) = temp_Q(\delta_D - 1) \vee (Appear_Q^-(\delta_D) \wedge \text{L\_shift}(Appear_X, |P''| - \delta_D - 1, 0))$;
$FT\_Appear_P^-(\delta_D) = Appear_Y \wedge \text{L\_shift}(\text{temp}_{P'}(\delta_D), 1, 0)$.

$FT\_Freq_{DSeq}^-(P)$ equals to the number of bits with value 1 in $FT\_Appear_P^-(\delta_D)$. Therefore the deletion fault-tolerant frequency of a pattern P could be counted efficiently to evaluate whether P is a FT-RP or not.

4 Mining Top-K Non-trivial FT-RPs with Min-length Constraint

In this section, two algorithms, called **TFTRP-Mine** and **RE-TFTRP-Mine,** are developed for finding **T**op-K non-trivial **FT-RP**s. These two algorithms are applicable for both situations considering insertion/deletion fault tolerance by exchanging the function of generating fault-tolerant appearing bit sequences.

4.1 TFTRP-Mine Algorithm

TFTRP-Mine Algorithm is designed based on the representation of fault-tolerant appearing bit sequences to mine top-K non-trivial FT-RPs. First, the data sequence is scanned once to create the bit index table. Initially, the candidate pattern is a single data item in the data sequence. If the candidate is a FT-RP, an additional data item is appended to the FT-RP to generate a longer candidate pattern. In other words, the candidate patterns are generated in *depth-firs* order. So the fault-tolerant appearing bit sequence of a candidate pattern is obtained according to the recursive function defined in the previous section. Then, the fault-tolerant frequency of a candidate pattern is counted efficiently to decide whether it is a FT-RP. According to the *anti-monotonic* property, it is not necessary to generate candidate patterns by appending additional data items to a non-FT-RP. Moreover, a FT-RP must satisfy the minimum length and non-trivial constraints before being outputted to the mining result. Finally, after sorting the mining result according to the fault-tolerant frequencies, the top-K non-trivial FT-RPs satisfying the *min_len* constraints are obtained from the first K patterns in the result. In summarizing the above descriptions, the mining process of TFTRP-Mine algorithm consists of the following steps.

Algorithm TFTRP-Mine:
Input: a data sequence *DSeq*, fault tolerance δ_I / δ_D, *min_len*, and *K*.
Output: Top-K non-trivial FT-RPs with length no less than *min_len*.
Step1. Scan *DSeq* once to construct the bit index table.
Let $D = \{D_1, D_2, \ldots D_n\}$denote the set of data items in *DSeq*.
Step2. Set *P* to be an empty data sequence. Set $l = 1$ and $j_l = 1$.

Step3. Generate longer candidate patterns:

Step3-1. Generate a new candidate P' by appending data item D_{jl} to *P*, and compute $FT\text{-}Appear_{P'}^{+}(\delta_I)$ or $FT\text{-}Appear_{P'}^{-}(\delta_D)$.

Step3-2. Count the number of bits with value 1 in $FT\text{-}Appear_{P'}^{+}(\delta_I)$ or $FT\text{-}Appear_{P'}^{-}(\delta_D)$ to get FT_freq_{DSeq}(P'). If $FT\_freq_{DSeq}(P') < min\_freq$, proceed to **Step3-6**.

Step3-3. Check whether *P'* satisfies the minimum length constraint. If $|P'| \geq min\_len$, insert *P'* into Minlen set.

Step3-4. Set $P=P'$, $l = l + 1$, $j_l = 1$, and recursively call **Step3**.

Step3-5. Check whether *P'*, is non-trivial by calling procedure **Non_Trivial**(*P'*, temporal results).

Step3-6. Set $j_l = j_l + 1$, If $j_l \leq$ n, proceed to **Step3-1**.

Step3-7. $l = l$-1. If $l > 0$, return the recursive call; otherwise, proceed to **Step4**.

Step4. Sort the temporal results in fault-tolerant frequency descending order. Extract the first *K* patterns from the temporal results.

If S is non-trivial among the patterns found until now according to the results in *Temp*, the procedure Non_Trieval(*S*, *Temp*) will insert S into *Temp*. Moreover, all the sub-patterns of S in Temp, which have the same frequencies with S, will be removed.

4.2 RE-TFTRP-Mine Algorithm

In TFTRP-Mine algorithm, all the FT-RPs in the data sequence are found first. Then, top-K non-trivial FT-RPs are extracted from the results. If huge amounts of FT-RPs exist, all FT-RPs still have to be mined out even only the top-K non-trivial FT-RPs need to be outputted. It causes the mining process costly even for a small K setting. Therefore, the refinement of TFTRP-Mine, RE-TFTRP-Mine algorithm is designed. In the refined algorithm, those FT-RPs which are not possible the top-K non-trivial FT-RPs are pruned as early as possible by raising *min_freq* during the mining process. The idea is to raise *min_freq* to be a higher value if the least frequency among the most updated top-K FT-RPs has became larger than *min_freq*. Then, the FT-RPs with fault-tolerant frequencies less than the new *min_freq* will not be used to generate longer candidate patterns in the following mining process. Moreover, in order to get the FT-RPs with high fault-tolerant frequencies as early as possible, among the FT-RPs which have been discovered, the FT-RPs with higher frequencies are assigned higher priorities used to generate new candidates.

The mining process of RE-TFTRP-Mine algorithm is shown below. Since the minimum length constraint is required, the two strategies described above are applied only after all the FT-RPs with length equal to *min_len* have been found and stored in Minlen_Heap. Then, the patterns in Minlen_Heap are sorted according to their fault-tolerant frequencies to decide their priorities for generating the following candidates.

Algorithm RE-TFTRP-Mine:

Input: a data sequence *DSeq*, fault tolerance δ_I / δ_D, *min_len*, and *K*.

Output: Top-K non-trivial FT-RPs with length no less than *min_len*.

Step1. Scan *DSeq* once to construct the bit index table.

Let $D = \{D_1, D_2, \ldots D_n\}$ denote the set of data items in *DSeq*.

Step2. Set *P* to be an empty data sequence. Set $l = 1$ and $j_l = 1$.

Step3. Generate longer candidate patterns:

Step3-1. Generate a new candidate P' by appending data item D_{jl} to *P*, and compute $FT\text{-}Appear_{P'}^{+}(\delta_I)$ or $FT\text{-}Appear_{P'}^{-}(\delta_D)$.

Step3-2. Count the number of bits with value 1 in $FT\text{-}Appear_{P'}^{+}(\delta_I)$ or $FT\text{-}Appear_{P'}^{-}(\delta_D)$ to get FT_freq_{DSeq}(P'). If $FT\_freq_{DSeq}(P') < min\_freq$, proceed to **Step3-5**.

Step3-3. Check whether *P'* satisfies the minimum length constraint. If $|P'| \geq min\_len$, insert P' into Minlen_Heap and proceed to **Step3-5**.

Step3-4. If $|P'| < min\_len$, set $P=P'$, $l = l + 1$, $j_l = 1$, and recursively call **Step3**.

Step3-5. Set $j_l = j_l + 1$. If $j_l \leq n$, go back to **Step3-1.**

Step3-6. $l = l - 1$, if $l > 0$, return the recursive call; else if $l = 0$, copy the *K* patterns in Minlen-Heap with top-K fault tolerant frequencies to temporal top-K set, and proceed to **Step4**.

Step4. Select a FT-RP to generate candidates:

Step4.1. Maintain the non-trivial FT-RPs patterns with Top-K fault tolerant frequencies in the temporal top-K set. Let *S* denote the pattern that has the least frequency among the Top-K patterns currently. If FT_freq_{DSeq}(S)> *min_freq*, set $min\text{-}freq = FT\_freq_{DSeq}$(S). Remove those patterns with fault-tolerant frequencies being less than the new *min_freq* from Minlen_Heap.

Step4.2. Set $P=\{Q|\ Q$ has maximum fault-tolerant frequency in Minlen_Heap$\}$ and remove *P* from Minlen_heap. Set $l=|P|$, $l = l+1$, $j_l=1$, and recursively call **Step3**.

Step4.3. Check whether P is non-trivial by calling procedure **Non_Trivial**(*P*, temporal top-K set).

Step5. Repeat **Step 4** until Minlen_Heap is empty.

Step6. Extract the first *K* patterns from the temporal top-K set to be the mining result.

5 Performance Study

We implemented TFTRP-Mine and RE-TFTRP-Mine algorithms using Borland C++ Builder 5.0. The experiments are performed on a 2.4GHz Intel Pentium IV PC machine with 512 megabytes main memory and running Microsoft XP Professional.

In the first five experiments, data sequences are produced from a synthesis data generator. Two input parameters are required when running the data generator, where *L* denotes the length and *E* denotes the number of various data items in the generated data sequence. The scalabilities of TFTRP-Mine and RE-TFTRP-Mine algorithms on execution time are compared under various parameters setting. Moreover, the results of mining repeating patterns in real music objects without fault tolerance and with fault tolerance are compared in the last experiment. According to theses experiment results, the effectiveness of mining with fault tolerance is observed.

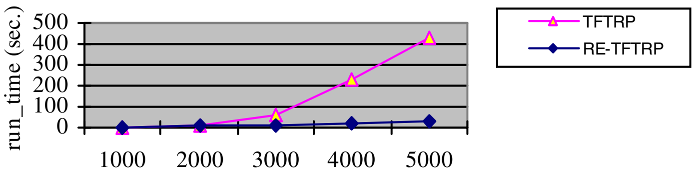
In addition to the data parameters *L* and *E* defined previously, δ_I(the insertion fault tolerance), *min_len*(the minimum length constraint) and *K*(the desired number of non-trivial FT-RPs to be mined) also influence the mining results and execution time of the proposed algorithms. By varying one of these five factors (L, E, δ_I, *min_len*, and K) in each experiment, the scalabilities of TFTRP-Mine and RE-TFTRP-Mine on execution time are observed. Besides, in order to show the pruning effect of RE-TFTRP-Mine, the numbers of generated candidate patterns of two algorithms are also illustrated. In the following five experiments, *min_freq* is fixed to be 10.

[Experiment 1] Changing the size of a data sequence (L)
In this experiment, δ_I = 2, *min_len* = 8, K = 5 and E = 5 are given. L is varied from 1000 to 5000. The experimental results in Fig. 1 show the execution efficiency of RE-TFTRP-Mine algorithm outperforms the one of TFTRP-Mine algorithm. The reason is that the former does not need to generate all candidate patterns when finding top-K non-trivial FT-RPs. Moreover, the number of generated candidate patterns increases as the value of L is raised. Therefore, when L increases, the execution efficiency of TFTRP-Mine is slower and slower than the one of RE-TFTRP-Mine algorithm.

[Experiment 2] Changing the number of various data items (E)
Fig. 2 shows the execution times of the proposed two algorithms on data sequences with L=2000, where δ_I=2, *min_len*=8 and K=5 are inputted. When E increases from 5 to 25, the generated candidate patterns also increases. Thus, the performance efficiencies of two algorithms decrease in this range. However, when E=30, the numbers of generated candidates in both algorithms become less than the ones generated when E=25 and the corresponding execution time of both algorithms is also lowered down. The reason is that more various data items may cause the data sequence becomes more “sparse”. Therefore, fewer FT-RPs are found and fewer candidate patterns are generated even there are more various data items.

[Experiment 3] Changing insertion fault tolerance (δ_I)
This experiment is performed on data sequences with L=2000, where E=5, *min_len*=8 and K=5 are inputted. As the results shown in Fig. 3, when δ_I increases, the number of generated candidate patterns grows exponentially because much more FT-RPs are found due to the relaxed constraints. Therefore, the execution time of two algorithms also increases as δ_I increases. However, RE-TFTRP-Mine still prunes huge amount of unnecessary candidates dramatically.

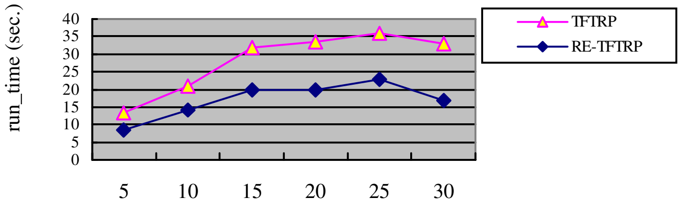
[Experiment 4] Changing the minimum length (*min_le*)
This experiment is performed on data sequences with L=2000 and E=5, where δ_I=2 and k=5 are inputted. For the same data sequence, no matter what value the *min_len* is, the number of generated candidate patterns in TFTRP-Mine algorithm is the same (41,730) and the curve of execution time keeps almost steady. On the other hand, RE-TFTRP-Mine algorithm finds all the FT-RPs with lengths equal to *min_len* before tuning the *min_freq*. Therefore, the number of generated candidates of algorithm increases as *min_len* increases. In addition, because the longer patterns usually have lower frequencies, the larger *min_len* is, the less number of non-trivial FT-RPs are discovered. Thus, the number of non-trivial FT-RPs in the data sequence is less than 5 when *min_len* = 45 and 50. It implies that the setting of *min_freq* was not raised during the execution of RE-TFTRP-Mine algorithm. In this situation, RE-TFTRP-Mine



L	1000	2000	3000	4000	5000
TFTRP	10295	41730	120075	348610	533770
RE_TFTRP	8705	24300	24760	36090	41280

Number of generated candidates

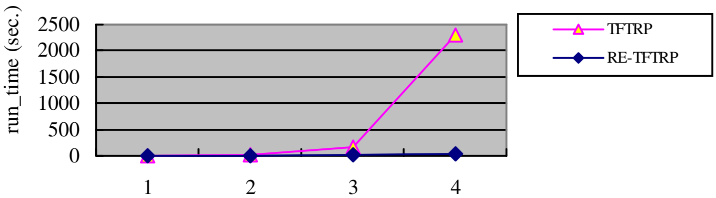
Fig. 1. Result of Experiment 1



E	5	10	15	20	25	30
TFTRP	41730	70840	103110	112325	120780	111408
RE_TFTRP	24300	38540	154825	56760	66140	46656

Number of generated candidates

Fig. 2. Result of Experiment 2



δ_I	1	2	3	4
TFTRP	5760	41730	394515	3434805
RE_TFTRP	5465	24300	36600	76195

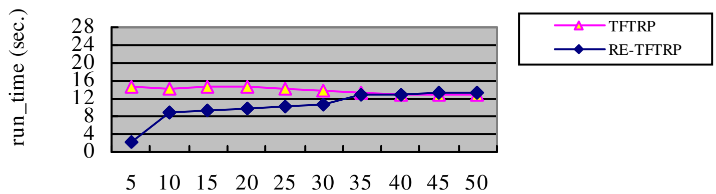
Number of generated candidates

Fig. 3. Result of Experiment 3

algorithm generates the same candidate patterns as TFTRP-Mine does and needs additional cost to maintain the sorted FT-RPs and the top-Ks. So the execution time of RE-TFTRP-Mine is over the one of TFTRP-Mine when min_len is 45 and 50.

[Experiment 5] Changing the setting value of *K*

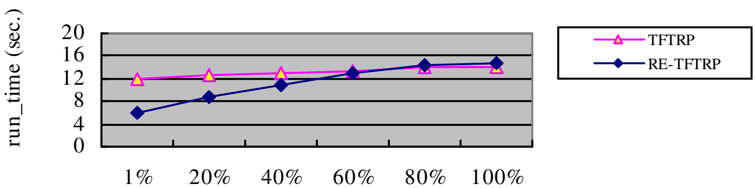
In this experiment, data sequences with L=2000 and E=5 are used as test data, where the run time parameters δ_I=2 and min_len=8 are given. Let max_K denote the number



min_len	10	20	30	40	50
TFTRP	41730	41730	41730	41730	41730
RE_TFTRP	25240	27615	30500	39620	41730

Number of generated candidates

Fig. 4. Result of Experiment 4



K/max_K	1%	20%	40%	60%	80%	100%
TFTRP	41730	41730	41730	41730	41730	41730
RE_TFTRP	14320	24300	31735	37325	41730	41730

Number of generated candidates

Fig. 5. Result of Experiment 5

of total non-trivial FT-RPs with *min_len* constraints discovered in this test data sequence. K is varied from *max_K*×1% to *max_K*×100%. Fig. 5.5 shows that the number of generated candidates in RE-TFTRP-Mine is the same with the one generated in TFTRP-Mine when K/max_K is more than 80%. This case occurs because the least-frequency in the top 80% non-trivial FT-RPs is the same with *min_freq*. Therefore, the pruning strategy does not work and more processing cost of RE-TFTRP-Mine is required than TFTRP-Mine. However, the execution time of RE-TFTRP-Mine is about half of the time of TFTRP-Mine because RE-TFTRP-Mine prunes about two third of the candidate patterns when K/max_K is 1%.

[Experiment 6] Performance evaluation on effectiveness
In this experiment, five popular songs are selected as test data, whose total playing-times are between 4 and 5 minutes. The run-time parameters *min_freq* = 3, K = 2 and *min_len* =8 are given. We compare the found repeating patterns under various setting of δ_I or δ_D with the actual motifs in the music object. The results show that no non-trivial FT-RPs satisfying the *min_len* constraint could be found among the five music objects if fault-tolerant mapping is not allowed. When at most two insertion/deletion errors are allowed (δ_I or $\delta_D = 2$), the found patterns are most close to the motifs in the music objects. It shows that mining repeating patterns with fault tolerance is necessary.

6 Conclusion and Future Works

In this paper, two algorithms, named TFTRP-Mine and RE-TFTRP-Mine, are proposed to mine top-K non-trivial fault-tolerant repeating patterns with lengths no less than minimum length constraints from data sequences. By extending the idea of appearing bit sequences, fault-tolerant appearing bit sequences are defined to represent the positions where candidate patterns appear in a data sequence with Insertion/deletion errors. Both of two algorithms use the recursive formulas to obtain fault-tolerant appearing bit sequences of a pattern systematically and then the fault-tolerant frequency of each candidate pattern could be obtained quickly. Besides, RE-TFTRP-Mine adopts two additional strategies to improve the mining efficiency. The experimental results show that RE-TFTRP-Mine outperforms TFTRP-Mine algorithm when *K* and *min_len* are small. In addition, when adopting fault tolerant mining, more important and implicit repeating patterns could be found for music objects.

References

1. C. C. Liu, J. L. Hsu, and A. L. P. Chen, "Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Databases," in Proceedings of the 15th International Conference on Data Engineering (ICDE'99), 1999.
2. J. Han, J. Wang, Y. Lu, and P. Tzvetkov, "Mining Top-K Frequent Closed Patterns without Minimum Support", in Proceedings of 2002 International Conference on Data Mining (ICDM'02), 2002.
3. J. L. Hsu, C. C. Liu, and A. L.P. Chen, "Efficient Repeating Pattern Finding in Music Databases," in Proceedings of the Seventh International Conference on Information and Knowledge Management(ACM CIKM'98), 1998.
4. J. L. Koh and W. D. C. Yu, "Efficient Feature Mining in Music Objects," Lecture Notes in Computer Science: DEXA'01: Database and Expert Systems Applications, pp. 221-231, Springer-Verlag, 2001.
5. J. L. Koh and P. W. Yo, "An Efficient Approach for Mining Fault-Tolerant Frequent Itemsets based on Bit Sequences," Lecture Notes in Computer Science: DASFAA'05: Database Systems for Advanced Applications, Springer-Verlag, 2005.
6. J. Pei, A. K.H. Tung, and J. Han, "Fault-Tolerant Frequent Pattern Mining: Problem and Challenges," in Proceedings of ACM-SIGMOD International Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'01), 2001.
7. P. Tzvetkov, X. Yan, and J. Han, "TSP: Mining Top-K Closed Sequential Patterns", in Proceedings of 2003 International Conference on Data Mining (ICDM'03), 2003.