FastPET：一快速粹取音樂資料中非不重要重覆片段之技術
FastPET: A Fast Non-trivial Repeating Pattern Extracting Technique for Music Data

羅有隆 游合成 范美琴
Yu-lung Lo *Ho-cheng Yu* *Mei-chin Fan*

朝陽科技大學 資訊管理系
台中縣霧峰鄉吉峰東路 168 號
Department of Information Management Chaoyang University of Technology
168,GiFeng E.Rd., WuFeng TaiChung County,Taiwan 413
Republic Of China
yllo@cyut.edu.tw *s8854618@cyut.edu.tw* *s8854608@cyut.edu.tw*

摘要

重覆片段為一串音符在音樂物件中重覆出現一次以上的音樂片段，而在一首音樂中會出現多次的片段，大部份均為主旋律或較容易為人們所熟悉與記憶的，因此我們可以利用此特徵來建構索引以加快音樂內容的搜尋。而非不重要重覆片段常用於分析音樂的重覆片段與尋找主旋律，所謂非不重要重覆片段，乃指在所有重覆片段中扣除該片段已全包含於其他更長片段中的項目，如此能大大減少重覆片段的項數，並大幅提昇音樂搜尋的效率。在這篇論文中，我們提出一種快速片段粹取技術，它能快速且有效率的在音樂資料中發現非不重要重覆片段，並且以實驗證明此方法確實較現行已提的方法，來得有效率。

關鍵詞: 音樂資料庫 重覆片段 非不重要重覆片段 內容的擷取

Abstract

A repeating pattern is a sequence of notes appearing more than once in a music object. Most of the repeating patterns are key melodies or easy to familiarize and remember for people. Therefore, we can use the themes or the repeating patterns to construct indices that can speedup music retrieval. A non-trivial repeating pattern is commonly used in analyzing the repeated part of music data and looking for themes. Non-trivial repeating patterns exclude those patterns which are all contained in other longer patterns, such that they can reduce the redundancy of the repeating patterns and speed up music search. In this paper, we proposed an approach called Fast Pattern Extracting Technique (FastPET) which can find all non-trivial repeating patterns in music objects fleetly and efficiently.

Keywords : music database, repeating pattern, non-trivial repeating pattern, content-based retrieval

一、 簡介

在過去的數年中，資料庫的研究人員已經發現，在未來將有越來越多非文字型式的資料需要被處理。 這些資料不像過去傳統僅有的數字或文字資料，未來資料庫中的新資料型態將包含有影像資料、聲音資料、影片資料、文件資料等等，為了操作這些新型態資料，已完成的研究工作，大多以內容擷取(content-based retrieval)來處理文件、影像與影片資料[4, 5, 12]。然而，聲音資料的擷取卻較少受到重視。所幸近來在多媒體資料庫中如何有效率的管理音樂資料，已獲得了較高度的注意[8, 11, 13, 16]。現在大部分的研究工作，多利用音樂資料的特性萃取(feature extraction)，例如：旋律、節拍、和弦等等，來建立索引，以加速音樂資料的搜尋。這些音樂資料特性大多可以轉化為字串型式來表示[8, 9, 13, 16]，再根據這些字串表示方式，發展有效率的字串索引(string indices)以幫助音樂資料的擷取(music data retrieval)，現已有許多的研究報告為字串的比對，提供了有效的解決方案[2, 14]。

最近以內容擷取(content-based retrieval)方式取得聲音及音樂資料的研究，已被大量的提出，Foote[6] 提出一個方法從音樂的訊號(signals)取得音樂的特徵，對於類似符合(similarity matching)，他使用一種樹狀結構量化的方式以降低電腦的運算。另外，Pfeiffer[15] 發展一組的工具以取得聲音的基音(fundamental frequency)以作為音樂的特徵，並以範例查詢(query-by-example)配合相

關運算的方式查詢資料，以計算音樂出現的次數。

最直接查詢音樂資料庫的方式，乃是哼唱一段音樂或由鍵盤輸入一段樂曲，做為查詢的範例以取得相似的音樂物件，而使用此種方法的系統已發表在[7, 11, 16]。Ghias[7]等人則發展一套從麥克風直接哼唱，並將音樂的訊號轉換成字串型態，且以'U'、'D'、'S'表示音符的'升音'、'降音'、'平音'，以允許任何音調的查詢。Tseng [16]也發展一套系統，架構在 WWW 上，允許使用者輸入簡譜做查詢或利用網路上的音樂編輯軟體以哼唱方式輸入查詢，甚至不需輸入任何音樂資料而選用系統提示的關鍵旋律進行「範例查詢」。清華大學陳良弼教授等人[3, 8, 9, 11, 13]所發展的系統，也以哼唱的方式作為查詢輸入的技術，另也運用和旋、節奏、旋律等音樂特性作為內容特徵與查詢條件，所採用的技術包括 1D-list 及 PAT-tree 等。

在以內容擷取為主的音樂查詢程序上，無論是以哼唱(humming)、旋律(melody)、和弦(chord)、節奏(rhythm)或字串等方式做為其核心技術，其效能均與音樂的長度息息相關，因此若音樂的長度過長則將嚴重影響查詢時間，甚至使人無法接受。在檢視音樂的組成上我們可以發現有很多一序列的音符結合在一起，其在整首音樂中出現超過一次以上者，我們稱此為重覆片段(repeating pattern)，例如在小蜜蜂這首歌中'sol-mi-mi-fa-re-re'重覆出現過三次，則'sol-mi-mi-fa-re-re'即為一重覆片段。在很多音樂學及音樂心理學的研究中也認同重覆片段在音樂結構中為一普遍性的特徵[1, 10]，因為重覆片段的長度較實際音樂短，因此若以重覆片段來表示實際音樂，則對以內容擷取式的音樂搜尋將會使其效能大大提昇。所以如何快速且有效的發現重覆片段，為本研究報告所要探討的問題。

在這篇文章的其他章節安排如下，我們在第二節將針對重覆片段在音樂資料中扮演的角色做一簡單的介紹，在第三節，針對現行產生重覆片段的技術做一簡單的概述，其中包含相關矩陣[8] 及 RP-TREE[9, 13]等方法，而在第四節中，我們提出一種叫做快速片段粹取技術(Fast Pattern Extracting Technique, FastPET)的演算法，它是由相關矩陣的基本觀念，改善其產生非不重要重覆片段的過程而來，它能提供較相關矩陣及 RP-TREE 更簡單、更快速的方法，以產生非不重要重覆片段，在第五節我們以實驗證實我們的方法確實較上述方法來的有效率，最後我們做一個總結並探討未來的研究方向。

二、 重覆片段在音樂資料中扮演的角色

在這節我們針對重覆片段及非不重要重覆片段其在音樂資料中所扮演的角色及其定義作一簡單的介紹，首先我們先對重覆片段及非不重要重覆片段做一簡單定義。

重覆片段(repeating pattern)：在 S 字串中的子字串出現次數大於 1 次以上且此子字串必須為 2 個字元以上，即必須扣除僅單 1 字元之項目，我們稱此字串為 S 字串中的 repeating pattern。

非不重要重覆片段(non-trivial repeating)：在 S 字串中的子字串 X 其出現次數，比任何包含他的較長字串的重覆次數之合為多[9, 13]。 亦即若子字串出現次數等於包含它的所有較長字串重覆次數之合，將不被記入。 一般而言非不重要重覆片段，大多為一首音樂中的主弦律或關鍵弦律，也是人們對一首歌，最不容易忘記的部份，因此我們可以藉由找出非不重要重覆片段，並建立索引以加快音樂資料的搜尋。

我們舉例說明非不重要重覆片段，假設有一 12 個音符的音樂字串'fcadcafadfca'，我們以 S 來代表此字串，對於 S 字串中的任一子字串 Y，假如其出現在 S 字串中的次數大於 1 者，我們稱此子字串為重覆片段，另該片段的長度則以|Y|來表示，*Freq(Y)*則代表該片段出現的次數[13]。

依據上述的說明我們可以知道，音樂字串'fcadcafadfca'的所有重覆片段如表 1 所列，但重覆片段中'a'， 'c'， 'd'， 'f'，因其僅有一個字元，對於音樂的索引並沒有太大的助益，因此我們將單一字元的重覆片段視為不重要重覆片段，另'fc'是'fca'的子字串且 *Freq('*fc'*) = Freq('*fca'*)* = 2，所以我們知道'fc'已包含於'fca'中，則'fc'是一個不重要重覆片段，而'fca'則為非不重要重覆片段。 另外，'ca'是'fca'的子字串但 *Freq('*ca'*)* = 3 而 *Freq('*fca'*)* = 2，'ca'出現的次數較'fca'為多，意謂著'ca'除了為'fca'的子字串外，必定還出現於音樂的其他位置，因此'ca'及'fca'均為非不重要重覆片段，最後可以在音樂字串'fcadcafadfca'中得到所有非不重要重覆片段有'ad'， 'ca'， 'fca'三個 。

表 1 音樂字串’fcadcafadfca’中所有重覆片段

Repeating Pattern	a	c	d	f	ad	ca	fc	fca
Frequency	4	3	2	3	2	3	2	2
Pattern Length	1	1	1	1	2	2	2	3

三、 現有技術

找尋非不重要重覆片段並做為音樂資料的索引，以加快音樂資料的搜尋，最近已有許多論文提出這方面的研究，而他們所採用的技術大都利用 suffix tree、矩陣及字串結合等方式。 Suffix tree 是一種字串索引結構，它可以表示出一個字串的所有字尾(suffix)[2, 14]，因所有字尾均存在 suffix tree 中，所以它也可以用來找出所有的重覆片段，但此種方法必須花費較多的儲存空間及執行時間，以找出重覆片段[9, 13]。 Tseng 於[16]中介紹所開發的音樂搜尋系統亦提出一種關鍵旋律取得(key melody extraction)的方法產生重覆片段，其乃利用相鄰音符的結合並計算其出現次數是否超過其所設定的門檻，若超過則屬重覆片段，依此方法由 2 個音符開始，接著 3 個音符，直到最大的重覆片段的長度為止，此種其運算複雜度較高。 Hsu 等人於[8]提出一種叫做相關矩陣(correlative matrix)，以及 Liu 等人於[13]提出一種方法則以建構 RP-TREE 的方式產生重覆片段，兩種方式均能快速的產生非不重要重覆片段(non-trivial repeating pattern)。 以下我們針對相關矩陣以及 RP-TREE 等方法做一簡單的概述。

(一) 相關矩陣(Correlative Matrix)

Hsu 等人在[8]提出相關矩陣的方法，以產生重要重覆片段，在這方法中它先依據音樂字串的長度建立一個上三角矩陣，再依不同情況紀錄各片段出現的次數及被其他片段所包含的次數，最後再過濾不重要重覆項目並重新計算該片段真正出現的次數。以下僅以一簡單例子說明。

假設有一段 12 個音符的音樂字串‘caaccaacdcbc’，以 S 來代表此段音符的字串，因為 S 字串中共有 12 個音符，所以相關矩陣必須先建立一個 12 × 12 的右上三角矩陣如圖 1，如果以 $T_{i,j}$ 表示此矩陣中各個格子的值，i 表示矩陣中第幾個列(row)，而 j 則表示矩陣中第幾個欄(column)，則矩陣內容建立方式為，若第 i 列所代表的符號與第 j 欄所代表的符號相同，如此 $T_{i,j}$ 的值可由 $T_{i,j} = T_{(i-1),(j-1)}+1$ 的規則求得，其建立結果顯示在圖 1。

	c	a	a	c	c	a	a	c	d	c	b	c
c	-			1	1			1		1		1
a		-	1			2	1					
a			-			1	3					
c				-	1			4		1		1
c					-			1		1		1
a						-	1					
a							-					
c								-		1		1
d									-			
c										-		1
b											-	
c												-

圖 1 處理完所有音符後的相關矩陣

在建構完相關矩陣之後，接下來就是要找出所有的重覆片段及重覆次數，在這使用了一種集合叫做 Candidate Set (以 CS 來表示)，以紀錄重覆片段及重覆出現次數，每一個 CS 包含有三個參數，其格式為(*pattern, rep_count, sub_count*)，其中的 *pattern* 代表重覆片段，而 *rep_count* 代表其出現的機率，而 *sub_count* 則表示此片段為其他片段子字串的次數，其最主要的作用乃在作為檢查此片段是否僅出現在另一片段之內(即 *rep_count* = *sub_count*)，此種情形，最後將不會被列出，因為它不是一個非不重要重覆片段。

這候選集合最初時，是一個空集合，對每一個在相關矩陣 T 中，當 $T_{i,j}$ 的元素非 0 項目，必須將其情況紀錄至候選集合中，根據 $T_{i,j} \geq 1$ 與 $T_{(i+1),(j+1)}$ 的條件關係，共有以下四種情況

Case 1: ($T_{i,j}$ =1 and $T_{(i+1),(j+1)}$ = 0)，例如，$T_{1,4}$=1，而 $T_{2,5}$ =0，代表’c’在這位置出現一次且其重覆片段的長度為 1，而 $T_{2,5}$=0 則表示其並沒有被其他片段所包含，所以我們將紀錄插入(‘c’,1,0)至 CS 中。

Case 2: ($T_{i,j}$ =1 and $T_{(i+1),(j+1)} \neq 0$)，例如，$T_{1,5}$=1，而 $T_{2,6} \neq 0$，則表示’c’為另一個重覆片段‘ca’的子字串，而且因為’c’已在 CS 中所以修改(‘c’,1,0)為(‘c’,2,1)以紀錄另一個’c’的片段被發現。

Case 3: ($T_{i,j}$ >1 and $T_{(i+1),(j+1)} \neq 0$)，例如，$T_{2,6}$=2，而 $T_{3,7}$ =3，而在此 $T_{2,6}$=2 表示有’ca’及’a’二個重覆片段，但因 $T_{3,7}$=3，表示’ca’及’a’二個重覆片段均為另一個重要片段’caa’的子字串，所以必須插入(‘ca’,1,1) 及 (‘a’,1,1) 至 CS 中。

Case 4: ($T_{i,j}$ >1 and $T_{(i+1),(j+1)}$ =0)，例如，$T_{4,8}$=4，而 $T_{5,9}$ =0，$T_{4,8}$ 是 4 代表有四個重覆片段‘caac’， ‘aac’， ‘ac’， ‘c’出現在這個位置，再者因為 $T_{5,9}$ =0，所以表示在這已沒有其他重覆的片段會包含‘caac’，所以必須插入(‘caac’,1,0)，(‘aac’,1,1)，(‘ac’,1,1)，並修改(‘c’,6,1)為(‘c’,7,2)。

在檢查所有非 0 元素之後，這候選集合 CS 應包含{('c',15,1), ('a',6,2), ('ca',1,1), ('caa',1,1), ('aa',1,1), ('caac',1,0), ('aac',1,1), ('ac',1,1)}，建構完所有集合的元素之後，有兩項重要的工作必須去執行，首先從 CS 中移除 *rep_count* = *sub-count* 項目，例如('ca',1,1), ('caa',1,1), ('aa',1,1), ('aac',1,1), ('ac',1,1) 均須被移除，因為它們都屬('caac',1,0)的子字串。第二，必須正確計算出 CS 中所有重覆片段的次數。在此他們以公式

$$f = \frac{1+\sqrt{1+8\times rep\_count}}{2}$$

以求得實際的出現次數 f，例如'c'的 *rep_count* = 15，代入公式後求得解為 6，所以'c'真正出現的次數為 6，以此方法即可求得所有的非不重要重覆片段及其實際出現的次數。最後產出所有的非不重要重覆片段如表 2

表 2 音樂字串'caaccaacdcbc'的所有非不重要重覆片段

Repeating Pattern	a	c	caac
Frequency	4	6	2
Pattern Length	1	1	4

(二) RP-TREE

Hsu 等人提出相關矩陣之後，又再提出 RP-TREE 演算法來產生非不重要重覆片段[9, 13]，此種方法乃利用字串結合(string-join)的方式，其做法大概可分為以下幾個步驟完成，

(1) 利用字串結合的方法，找出所有以 2 的 k 次方(2^k, $k \geqq 0$ 且 $2^k \leqq$ S)為級數的所有字串，並將該字串置於字串集合中，而此字串集合的格式為{X, *Freq(*X*)*,(*position_1*, *position_2*, …)}，其中 X 為 S 字串中的一個片段，而 *Freq(*X*)*代 X 片段在 S 字串中出現的次數，(*position_1*, *position_2*, …)則代表該片段在 S 字串出現的位置
(2) 找出最長的重覆片段長度
(3) 建構 RP-TREE
(4) 剔除 RP-TREE 中不重要重覆片段
(5) 找出所有非 2 的次方長度的片段並重建 RP-TREE
(6) 產生所有非不重要重覆片段

以下我們以一個實例來說明，假設有一個音樂特徵的字串 S='abcdefghabcdefghijabc'，首先在 S 字串中找出 $2^0 = 1$ 的字串結合，這也就是產生所有長度為 1 的字串集合，其結果如下：

RP[1] = {{'a', 3, (1, 9, 19)}, {'b', 3, (2, 10, 20)}, {'c', 3, (3, 11,21)}, {'d', 2, (4, 12)}, {'e', 2, (5, 13)}, {'f', 2, (6, 14)}, {'g', 2, (7,15)}, {'h', 2, (8, 16)}}

其中{'a', 3, (1, 9, 19)}乃表示字串'a'，於 S 字串中出現 3 次，分別出現於第 1,9,19 字元的位置，餘此類推。 然後利用 RP[1]的結果，將各字元位置相鄰的字串結合產生長度為 2 的字串，其結果如下：

RP[2] = {{'ab', 3, (1, 9, 19)}, {'bc', 3, (2, 10, 20)}, {'cd', 2,(3, 11)}, {'de', 2, (4, 2)}, {'ef', 2, (5, 13)}, {'fg', 2, (6, 14)},{'gh', 2, (7, 15)}}

利用同樣的方法可以產生 RP[4]及 RP[8]，其結果如下：

RP[4] = {{'abcd', 2, (1, 9)}, {'bcde', 2, (2, 10)}, {'cdef',2, (3, 11)}, {'defg', 2, 4, 12}}, {'efgh', 2, (5, 13)}}

RP[8] = {{'abcdefgh', 2, (1, 9)}}

接下來步驟，要找出最長的重覆片段，因為在 S 字串中已無比長度 8 更長的片段，所以 RP[8]已為其最長的片段，圖 2 顯示 RP[1]-RP[8]所建構 RP-TREE 的結果。

雖然在上述的方法中可以找出 2^0,2^1..2^k 的片段，但我們還是必須要去檢查是否還有其他長度的重覆片段存在，再者我們要的是非不重要重覆片段，對於那些不重要的片段(即被其他字串完全包含)，應先予以刪除，在圖 3 中為剔除 2^k 次方中所有不重要重覆片段之後的結果。

在刪除所有 2^k 次方的所有不重要重覆片段之後，我們必須要去檢查是否還有其他長度的重覆片段存在，在圖 4 中即顯示出由字串長度 2，結合而成字串長度為 3 的重覆片段，接下來必須再檢查是否還有其他長度的重覆片段存在，待全部檢查完之後，再剔除其他不重要重覆片段，最後在表 3 顯示執行完 RP-TREE 後之所有非不重要重覆片段。

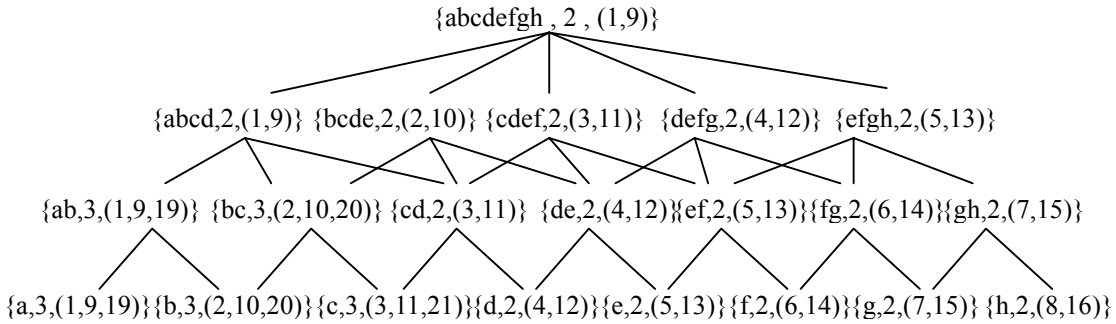

圖 2 音樂特徵 S 字串的 RP-TREE

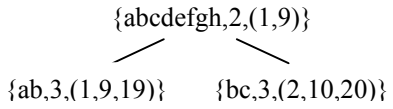

圖 3 刪除所有長度為 1、2、4 中不重要重覆片段後的 RP-TREE

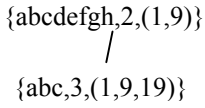

圖 4 刪除所有不重要重覆片段後的 RP-TREE

表 3 音樂字串'abcdefghabcdefghijabc'之所有非不重要重覆片般

Non-trivial repeating pattern	abc	abcdefgh
Frequency	3	2
Pattern Length	3	8
Starting position	1,9,19	1,9

四、快速片段粹取技術(FastPET)

在這節中介紹我們所提出之快速片段粹取技術(Fast Pattern Extracting Technique, FastPET)的演算法，並以實際的例子來說明如何利用此方法產生非不重要重覆片段與其出現之次數。

(一) FastPET 演算法

快速片段粹取技術的演算法大概分為以下七個步驟完成：

(1) 仿照三、(一)節將音樂字串建構相關矩陣，假設我們以 M 代表此矩陣，$M_{i,j}$ 表示矩陣中的位置值，此外以 S_j 表示字串中第 j 個字元，除 $i \geqq j$ 時之 $M_{i,j}$ 不予考慮之外，在執行矩陣 $M_{i,j}$ 值設定時，此右上三角矩陣的所有 $i < j$ 的 $M_{i,j}$ 值設定如下：

$$\begin{cases} \text{For } S_i \neq S_j, M_{i,j} = 0 \\ \text{For } S_i = S_j, \text{if } i = 1 \text{ then } M_{1,j} = 1 \\ \qquad\qquad \text{else } M_{i,j} = M_{(i-1),(j-1)} + 1 \end{cases}$$

(2) 於第 i 列中依序尋找 $M_{i,j} = 1$ (若 i=1，則 $M_{i,j} = -1$ 亦接受)，而 $M_{(i+r-1),(j+r-1)} \geq 2$ 且($M_{(i+r),(j+r)} = 0$ 或($j+r-1$)=|S|)，其中 $r \geq 2$，則從 S_j 開始有一長度為 r 的重覆片段。

(3) 檢查如果 $M_{1,j} \neq -1$，執行步驟(4)；但若 $M_{1,j} = -1$，表示已有從 S_j 開始的非不重要重覆片段記載於集合陣列 P[|S|]中(P 記載著字串每一位置產生非不重要重覆片段的長度記錄)。 此時檢查是否 $P[j]$集合中已有步驟(2)所得長度 r 之重覆片段記錄，若有則忽略此片段，重回步驟(2)，反之，執行步驟(4)。

(4) 檢查矩陣第 $i+r-1$ 列(row)中否有其他大於等於 r 的值出現，如果有且假設位於 $M_{(i+r-1),h}$ ，亦即第 h 欄位

置 $M_{(i+r-1),h} \geq r$，則將所有 $M_{1,h-r+1} \neq -1$ 設定為 $M_{1,h-r+1} = -1$，且同時設定集合陣列 $P[h-r+1] = P[h-r+1] \cup \{r\}$。 並計算 h 發生次數，這其中 h 至少有一值為 $h = j+r-1$(或者說 $j = h-r+1$)。

(5) 檢查矩陣第 $i+r-1$ 欄(column)中是否有其他大於等於 r 的值出現，計算其次數，並累計於步驟(4)中，如此步驟(4)與(5)所總計的重覆次數加 1 (片段第一次出現位置，發生於對角線 $i = j$ 處)，即為所找到非不重要重覆片段於字串中的出現次數。

(6) 將此非不重要重覆片段與其出現的次數，記載於片段集合(PatternSet)中(步驟(3)中建立集合陣列 P 的目的，即在於避免同一片段重覆的加入片段集合中)。 針對矩陣中每一列循序執行(2)至(6)步驟。

(7) 輸出片段集合中之所有非不重要重覆片段與其出現的次數。

以下以 C 程式語言的虛擬碼來表示我們的演算法：

```
FastPET (S)
S: Input string;
{
   CreateMatrix(S);          /*  步驟(1) */
   for (i=1; i<=|S|; i++) {
       for (j=i+1; j<=|S|; j++) {
           if   (M[i,j] == 1 || (i=1 && M[i,j]=-1)){    /*  步驟(2) */
               r = Find_Rep_Pattern(M[i,j]);
               if   (r >=2 ) {
                  if   ((M[1,j] =! -1) || (Find_Pattern_in_P(S[j], r) == FALSE)){ /*  步驟(3) */
                       Row_Count = Check_Row(M, i+r-1, P);              /*  步驟(4) */
                      Column_Count = Check_Column(M, i+r-1);          /*  步驟(5) */
                      Add_to_PatternSet(S[j], r, Row_Count+Column_Count, PatternSet);
                                                                              /*  步驟(6) */
                   }
               }
           }
       }
   }
   Output(PatternSet);   /*  步驟(7) */
}
```

(二) 範例說明

現在以例子來說明，假設我們有一個 14 個音符(notes)的樂曲，以字串表示為 'abcdbcdabcabcd'，若 S 代表此音樂字串，則 S_4 即表此字串的第 4 個字元，而在此例中 S_4='d'，執行快速片段粹取技術時，首先必須建立一個 14×14 的右上三角矩陣($M_{i,j}$，$i<j$)，如圖 5 非陰影部分。

	a	b	c	d	b	c	d	a	b	c	a	b	c	d
a	-													
b		-												
c			-											
d				-										
b					-									
c						-								
d							-							
a								-						
b									-					
c										-				
a											-			
b												-		
c													-	
d														-

圖 5 音樂 S 字串的初始矩陣

再來我們依步驟(1)中的設定規範，一列一列的設定右上三角矩陣中的值，在這個例子中，第 1 列因 S_1 字元為'a'，而在第 8 及第 11 欄中的 S_8 及 S_{11} 其字元亦均為'a'，故在 $M_{1,8}$、$M_{1,11}$ 將其值設為 1，其執行結果顯示在圖 6。

	a	b	c	d	b	c	d	a	b	c	a	b	c	d
a	-							1			1			
b		-												
…														
…														
c													-	
d														-

圖 6 設定第一個音符後的矩陣

而在第 2 列中，其 S_2 字元為'b'， 而第 5 欄 S_5 其字元亦均為'b'，但 $M_{1,4} = 0$，所以 $M_{2,5} = M_{1,4}+1=1$；另第 9 欄 S_9='b'，且 $M_{1,8}=1$，故 $M_{2,9}= M_{1,8}+1=2$。 同樣的第 12 欄 S_{12}='b'，$M_{2,12} = M_{1,11}+1 = 2$。 餘此類推，可一一的算出矩陣中每一位置的值，在圖 7 中顯示整個建置後之矩陣。

	a	b	c	d	b	c	d	a	b	c	a	b	c	d
a	-							1			1			
b		-			1				2			2		
c			-			2				3			3	
d				-			3							4
b					-				1			1		
c						-				2			2	
d							-							3
a								-			1			
b									-			2		
c										-			3	
a											-			
b												-		
c													-	
d														-

圖 7 處理完所有音符後的矩陣

接下來執行步驟(2)，逐列檢查矩陣右上三角的所有設定值，是否有符合 $M_{i,j}$=1(或 i=1 且 $M_{i,j} = -1$)，且 $M_{(i+r-1),(j+r-1)} \geq 2$ 的項目。 首先在第 1 列中，可以發現在 $M_{1,8} = 1$，而 $M_{2,9} \geq 2$ 且 $M_{3,10} = 3$，可知有一長度為 3 之非不重要重覆片段從 S_8 位置開始，即為'abc'。 再來步驟(3)的檢查，由於 $M_{1,8} \neq 1$，繼續執行第(4)步驟。 檢查第 1+3-1(=3)列中所有長度大於等於 3 的位置，如圖 8 中橫虛線路逕所示，除了 $M_{3,10} = 3$ 外，尚有 $M_{3,13} = 3$，共 2 次，設定 $M_{1,10-3+1} = M_{1,8} = -1$ 與 $M_{1,13-3+1} = M_{1,11} = -1$，以及 $P[8] = P[8] \cup \{3\}$與 $P[11] = P[11] \cup \{3\}$。 接著執行步驟(5)，計算第 1+3-1(=3)欄中所有長度大於等於 3 的次數，如圖 8 中直虛線路逕所示，結果是 0 次，則'abc' 非不重要重覆片段共出現 2+0+1=3 次。 最後於步驟(6)中，將{'abc', 3}加入片段集合，PatternSet = PatternSet ∪ {'abc', 3}，結果顯示於圖 9 中。

	a	b	c	d	b	c	d	a	b	c	a	b	c	d
a	-							1			1			
b		-			1				2			2		
c						2				3			3	
d				-			3							4
..														
…														
d														-

'abc'

計算次數路徑

圖 8 找出'abc'的非不重要重覆片段及次數

	a	b	c	d	b	c	d	a	b	c	a	b	c	d
a	-							-1			-1			
b		-			1				2			2		
c						2				3			3	
d				-			3							4
b					-				1			1		
…														
…														
d														-

$P[8] = \{3\}, P[11] = \{3\}$
PatternSet = {{'abc',3}}

圖 9 找尋完'abc'片段後之結果

回到步驟(2)繼續執行，再次發現 $M_{1,11} = -1$ 且 $M_{4,14} = 4$ 已達矩陣盡頭(欄位 14=|S|)，可知有一長度為 4 之非不重要重覆片段從 S_{11} 位置開始，即為'abcd'。 執行步驟(3)時，雖然 $M_{1,11} = -1$，但 $P[11]$裡尚未有片段長度 4 之記錄，所以繼續往下執行步驟(4)。 除設定 $P[11] = P[11] \cup \{4\}$外，也由圖 10 的虛線路徑發現到，片段'abcd'於步驟(4)與(5)中出現 1+0+1=2 次。 最後於步驟(6)中，將{'abcd', 2}加入片段集合中，如圖 10 所示。

	a	b	c	d	b	c	d	a	b	c	a	b	c	d
a	-							-1			-1			
b		-			1				2			2		
c			-			2				3			3	
d							3							4
b					-				1			1		
…														
…														
d														-

$P[8] = \{3\}, P[11] = \{3, 4\}$
PatternSet = {{'abc',3},{'abcd', 2}}

圖 10 找尋完'abcd'片段後之結果

繼續執行，於第 2 列可以發現 $M_{2,5} = 1$ 且 $M_{4,7} = 3$，符合非不重要重覆片段的條件，找到片段'bcd'，長度為 3。於圖 11 的橫虛線路徑，可發現除 $M_{4,7} = 3$ 外，$M_{4,14} = 4$ 也大於長度 3，依照演算法，必須設定 $M_{1,5} = -1$ 與 $M_{1,12} = -1$，以及 $P[5] = P[5] \cup \{3\}$與 $P[12] = P[12] \cup \{3\}$。 而片段總共出現次數為 3。最

後將{‘bcd’, 3}加入片段集合中，其結果如圖 11 所示。

	a	b	c	d	b	c	d	a	b	c	a	b	c	d
a	-				-1			-1			-1	-1		
b		-			1				2			2		
c			-			2				3			3	
d				-			3							4
b					-				1			1		
…														
…														
d														-

P[5] = {3}, *P*[8] = {3},*P*[11] = {3, 4},
P[12] = {3}
PatternSet = {{‘abc’,3}, {‘abcd’, 2},
{‘bcd’, 3}}

圖 11 找尋完‘bcd’片段後之結果

同樣方法再往下執行，於 $M_{5,9}$ 處可得一從 S_9 開始之非不重要重覆片段‘bc’，長度為 2，次數為 4。而在 $M_{5,12}$ 處雖也發現一長度為 3 的重覆片段，但因 $M_{1,12}$ = -1 且 *P*[12]已有相同長度之記載，表示該片段已記錄過，所以必須忽略。又最後於 $M_{8,11}$ 處雖也發現一長度為 3 的重覆片段，但因 $M_{1,11}$ = -1 且 *P*[11]也有相同長度之記載，故忽略之。整個矩陣執行完畢，其結果顯示在圖 12，而表 4 則顯示出執行完快速片段粹取技術之後的所有結果。

	a	b	c	d	b	c	d	a	b	c	a	b	c	d
a	-				-1			-1			-1	-1		
b		-			1				2			2		
c			-			2				3			3	
d				-			3							4
b					-				1			1		
c						-				2			2	
d							-							3
a								-			1			
b									-			2		
c										-			3	
a											-			
b												-		
c													-	
d														-

P[5] = {3}, *P*[8] = {3},*P*[9] = {2}, *P*[11] = {3, 4},*P*[12] = {3}
PatternSet = {{‘bc’, 4}, {‘abc’,3}, {‘bcd’, 3}, {‘abcd’, 2}}

圖 12 執行完快速片段粹取技術後之結果

表 4 音樂字串’abcdbcdabcabcd’中所有非不重要重覆片段

Non-trivial repeating pattern	bc	abc	bcd	abcd
Frequency	4	3	3	2
Pattern Length	2	3	3	4
Starting position	2,5,9,12	1,8,11	2,5,12	1,11

五、實驗

為了檢驗新演算法的效能，在這節中我們對相關矩陣、RP-TREE、以及快速片段粹取技術的演算法做一系列的速度評估與比較。我們針對三個演算法各別設計程式，並於 PIII 700 的 Dell 電腦上執行音樂物件，以找出非不重要重覆片段，且記錄執行所耗費的時間。同時我們採用類似於[8, 9 , 13]的實驗方法，使用二組音樂資料集合，一組為人造的音樂資料(synthetic music data)，音符的分配可能較為均勻，且可依實驗需要而設計；而另一組則以實際的音樂資料(real music data)為範本。然而，會影響實驗結果的因素，大致可分為

(1) 音樂物件的大小
(2) 非不重要重覆片段長度之長短
(3) 非不重要重覆片段數目之多寡

以下將分別以此三項因素做實驗，以比較在不同條件下三個重覆片段演算法表現差異。

(一) 音樂物件大小的影響

在音樂物件大小影響的實驗中，我們以人造音樂資料做為實驗的資料，而實驗音樂物件的長度以及最長非不重要重覆片段的平均長度，列於表 5 中。分別執行各演算法，統計產生非不重要重覆片段所需耗費的時間，結果呈現於圖 13 中。在圖 13 中可以發現，快速片段粹取技術明顯優於其它兩種演算法，其尋找非不重要重覆片段所需耗費的時間一直是三者間最少的。相關矩陣演算法在音樂物件長度為 300 之後即沒有出現在圖中，此乃因為其執行的時間已超過圖中所設定的範圍。而 RP-TREE 的演算法優於相關矩陣，亦呼應了[9, 13]中的實驗結果，但 RP-TREE 仍然較快速片段粹取技術有所不及，隨著音樂物件長度的增加，其所需的執行時間亦大幅度的增加，與快速片段粹取技術的差距也就越來越大。當音樂物件長度達到 1000 個音符時，其所耗費的執行時間約是快速片段粹取技術的五倍。探究其原因，乃是 RP-TREE 的演算法可以快速的處理長度剛好為 2^k 的重覆片段，但卻必須花較長的時間處理長度非 2^k 的重覆片段，當音樂物件長度大幅增長後，非 2^k 的重覆片段，相對的亦大幅的增加，以至於必須耗費較長的執行時間。反觀圖中快速片段粹取技術的曲線，雖也是隨音樂物件長度的增加而增加，但卻是緩和許多，而近乎為上升直線。

表 5 實驗音樂物件長度與其最長非不重要重覆片段的平均長度

Object size	100	200	300	400	500
Average length of the longest repeating pattern	29	41	54	86	147
Object size	600	700	800	900	1000
Average length of the longest repeating pattern	186	187	198	212	220

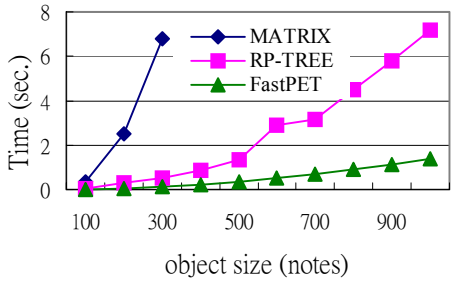

圖 13 Elapsed time vs. object size of music object

(二) 非不重要重覆片段長度之影響

此節實驗所指之非不重要重覆片段長度，為音樂物件中，最長之非不重要重覆片段的長度，由於它影響搜尋時間甚巨，是以必須對它做一番探討。我們測試最長非不重要重覆片段的平均長度為 8~256，而真實音樂物件資料長度隨最長非不重要重覆片段的長度呈遞增，平均為 55~623，而人造音樂物件資料長度則平均為 20~600。 真實音樂物件資料與人造音樂物件資料的實驗結果，分別表示於圖 14 與圖 15。 整體來說，三個演算法在真實音樂物件資料中，表現不如人造音樂物件資料，那是因為真實音樂資料較難有剛好長度是實驗值的 8、16、32、… 等等的最長非不重要重覆片段，且需要較長的音樂物件資料才能達到所求。反之，人造資料則為人為設計，且符合實驗的需求。 但是兩種不同資料實驗的結果，其曲線圖仍然有著類似的走勢。分析兩圖，同樣的，相關矩陣演算法表現的最差，而 RP-TREE 演算法仍然優於相關矩陣，再次的呼應了[9, 13]中的實驗結果。然而，RP-TREE 對於最長非不重要重覆片段長度大於 32 以後，執行耗費時間大幅的增加，亦即效率大幅滑落，理由同五、(一)節的說明。三個演算法都受到最長非不重要重覆片段長度的影響，然而對快速片段粹取技術的影響卻是最小。從圖中我們仍然發現到，相較於其它兩演算法，快速片段粹取技術依然表現得最佳，其曲線是隨最長非不重要重覆片段的長度很緩和的往上走。

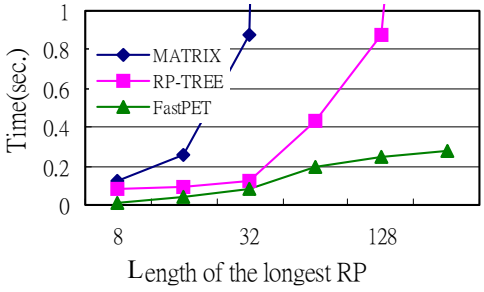

圖 14 Elapsed time vs. length of the longest repeating pattern for the real data set.

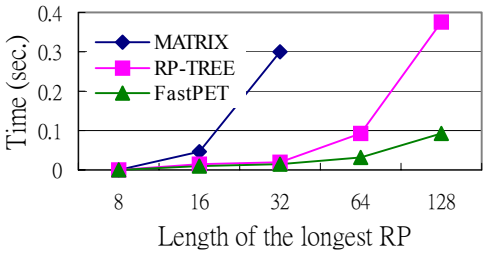

圖 15 Elapsed vs. length of the longest repeating pattern for the synthetic data set.

(三) 非不重要重覆片段數目的影響

非不重要重覆片段數目之多寡同樣的影響找尋非不重要重覆片段的速度，這節實驗就在於評估其對三個演算法的影響。 我們以人造音樂物件資料來做實驗，非不重要重覆片段的數目變化平均為 25~200，最長非不重要重覆片段的長度平均不超過 20，而所對映的音樂物件長度變化平均為 100 ~ 700。 圖 16 展示出實驗的結果。 在這實驗中可以看到這三種演算法，隨著非不重要重覆片段數目之增加，其耗費的執行時間亦是呈曲線上揚。快速片段粹取技術表現得仍然最好，其曲線遞增得最緩和。而沒有意外的，表現其次的是 RP-TREE 演算法，最差的仍然是相關矩陣演算法。 在此，RP-TREE 與快速片段粹取技術之間的差距，並未如前兩節的實驗結果，隨曲線的上升而大幅拉大，這是因為我們為了單純的凸顯非不重要重覆片段數目的影響，而將最長非不重要重覆片段的平均長度限制在 20 以下的結果。

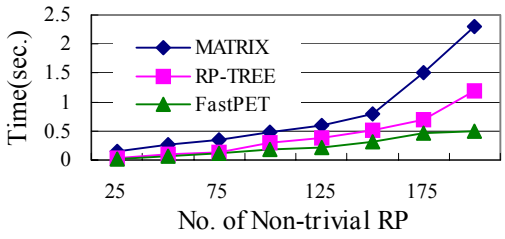

圖 16 Elapsed time vs. no. of non-trivial repeating pattern for the synthetic data set.

六、 結論與未來工作

在這篇研究報告中，我們提出了一種快速片段粹取技術的演算法，它能快速且有效率的搜尋出音樂物件資料中所有非不重要重覆片段，這些非不重要重覆片段往往為音樂之主旋律所在，若以此建立音樂資料庫之索引，將能幫助從龐大的音樂資料庫中更有效率的搜尋音樂。 我們的實驗亦證實了，快速片段粹取技術之執行效率，優於現有之相關矩陣以及 RP-TREE 演算法。 我們究其原因是，相關矩陣有重覆比對已產生之重覆片段集合之問題，而 RP-TREE 相當耗時於重覆片段長度不為 2^k 的搜尋。反之，快速片段粹取技術避免掉該兩演算法的缺點，使得執行效率優於它們。

以矩陣來尋找重覆片段，需要佔用較大之電腦記憶體空間來做運算，它所需記憶體空間為音樂資料長度的平方，複雜度雖高，但是音樂資料長度多為數百個音符以內，以目前的電腦硬體設備是足以適用的。然而，如果演算法可以更省記憶空間的執行運算，則除了可以運用在音樂資料之外，也可應用在找尋 DNA 中的重覆序列，人類 DNA 序列的長度達三十億，無法於現有電腦做矩陣的運算，這將是我們未來嘗試解決的方向。

七、參考文獻

[1] V. Bakhmutova, V. D. Gusev, and T. N. Titkova, 'The search for adaptations in song melodies,' *Computer Music Journal*, vol. 21, no. 1,

[2] M.T. Chen and J. Seiferas, 'Efficient and Elegant Subword-Tree Construction,' *In A. Apostolico and Z. Galil. Editors, Combinatorial Algorithms on Words*, Vol. 12 of NATO ASI Series F: Computer and System Sciences, Pages 97-107, Springer-Verlag, Berlin, Germany, 1984.

[3] Arbee L.P. Chen, M. Chang, J. Chen, J.L. Hsu, C.H. Hsu, and Spot Y.S. Hua 'Query by Music Segments: An Efficient Approach for Song Retrieval,' *In Proc. of IEEE Int'l Conf. on Multimedia and Expro*, 2000.

[4] G. Davenport, T.A. Smith, and N. Pincever, 'Cinematic Primitives for Multimedia,' *IEEE Computer Graphics & Applications*, Pages 67-74, July 1991.

[5] Y.F. Day, S. Pagtas, M. Iino, A. Khokhar, and A. Ghafoor, 'Object-Oriented Conceptual Modeling of Video Data,' *In Proc. Of IEEE Data Engineering*, Pages 401-408, 1995.

[6] J. Foote, 'Anoverview of audio information retrieval,' *Multimedia Systems*, vol. 7, no. 1, pp. 2-10, ACM Press & Springer-Verlag, 1999.

[7] A. Ghias, J. Logan, D. Chamberlin, and B.C. Smith, 'Query by Humming: Musical Information Retrieval in an Audio Database,' *In Proc. of ACM Multimedia*, Pages 231-236, 1995.

[8] J.L. Hsu, C.C. Liu, and Arbee L.P. Chen, 'Efficient Repeating Pattern Finding in Music Databases, ' *In Proc. of ACM Int'l Conf. on Information and Knowledge Management*, 1998.

[9] J.L. Hsu, C.C. Liu, and Arbee L.P. Chen, 'Discovering Non-Trivial Repeating Patterns in Music Data,' *IEEE transaction on Multimedia,* to appears.

[10] C.L. Krumhansl, *Cognitive Foundations of Musical Pitch*, Oxford University Press, New York, 1990.

[11] C.C. Liu, J.L. Hsu, and Arbee L.P. Chen, 'An Approximate String Matching Algorithm for Content-Based Music Data Retrieval,' *In Proc. of IEEE Int'l Conf. on Multimedia Computing and Systems*, Pages 451-456, 1999.

[12] C.C. Liu and Arbee L.P. Chen, '3D-List: A Data Structure for Efficent Video Query Processing,' *IEEE TKDE*, to appear.

[13] C.C. Liu, J.L. Hsu, and Arbee L.P. Chen, 'Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Databases,' *In Proc. of IEEE Data Engineering*, 1999, pp.14-21.

[14] E. McCreight, 'A Space-Economical Suffix Tree Construction Algorithm,' *Journal of Association for Computing Machinery*, Pages 262-272, 1976.

[15] S. Pfeiffer, S. Fischer, and W. Effelsberg, 'Automatic audio content analysis,' *In Proc. of ACM Multimedia Conference*, 1996, pp. 21-30.

[16] Y.H. Tseng, 'Content-Based Retrieval for Music Collections,' *In Proc. of ACM SIGIR'99*, Pages 176-182, 1999, Berkley, CA, USA.