以近似字串比對做為音樂資料允許容錯查詢之探討

羅有隆
朝陽科技大學
yllo@cyut.edu.tw

黃健錡
朝陽科技大學
s9714628@cyut.edu.tw

摘要

在音樂內容擷取(music content-based retrieval)的研究領域，大多都是利用特徵的萃取，例如旋律、節拍、和弦等等，再利用這些特徵建立索引，以加速對音樂資料的搜尋。而有關音樂資料查詢最直接且簡便的方式，乃是哼唱一段旋律或由鍵盤輸入一段樂曲，做為查詢樣本，分析出其音樂特徵，以尋找最近似的音樂物件。然而並非每位使用者都具有音樂專長，因此往往無法提供完全準確的音樂範例以進行精確查詢。因此，我們必須容許使用者在記憶中與現實音樂片段帶有些許差異的產生，如多音、少音與錯音等等。 也就是對音樂的查詢，必須具有允許容錯之近似搜尋的能力。目前於音樂資料中的允許容錯查詢仍然還有很大的改善空間，本研究的目的在於透過檢視現有字串比對技術，以尋找適用於音樂查詢所可能面臨之多音、少音、升音與降音等之近似比對的方法，期望再經過我們的改良，可以提升允許容錯之音樂查詢能力。

關鍵詞：音樂資料庫、內容擷取、字串比對、近似搜尋、允許容錯。

Abstract

In the researches of music content-based retrievals, the researchers extract the features, such as melodies, rhythms and chords, from the music data and develop indices that will help to retrieve the relevant music quickly. The query by example (QEB) is widely used for multimedia content-based retrieval. Since users are not all expert in music, we cannot expect users to be able to precisely specify music query examples. Therefore, certain errors in music query examples, such as dropout notes, insertion notes, and transposition notes, should be allowed for users' queries. That is, the fault tolerance ability in approximate search is needed for music query. However, there is still room for improvement in approximate music searching. In this research, we will investigate the ability of existing similar string matching techniques for performing the approximate music search. Thereafter, a new approach will be developed to enhance the fault tolerant music query.

Keywords: music database, content-based retrieval, string match, approximate search, fault tolerance

1. 前言

在資訊爆炸的時代，多媒體資料的發展相對的快速，與一般資料有很大的不同。多媒體資料的特徵多，例如：影像與影片的物件相對位置、色彩配置、動作，聲音的音調、旋律等等，並非一般的純文字與數字的資料庫所能比擬的。早期最常使用的為採用字串的關鍵字查詢，如:影片名稱、歌曲的作者、演唱者、歌曲名…等。然而，對於使用者而言，印象最深刻的，並不一定是這些關鍵字，而是多媒體資料的一些小片段，如:電影的場景、歌曲的旋律、部份歌詞。近幾年，對於內容為主的多媒體資料擷取(content-based multimedia data retrieval)的研究越來越多[7][10][8][22][22]。在音樂資料庫中，對於內容的擷取，主要是將音樂中的旋律(Melody)、節拍(Rhythm)、和弦(Chord)等特徵擷取出來，轉化為字串的型式表示[15][13][3][12]，並利用這些特徵所轉化而成的字串建置成索引，來進行有效率的音樂資料擷取(music data retrieval)。而在音樂資料庫的字串比對上，也已有許多的研究報告提出了有效率的解決方法[4][20][25]。

在音樂資料的查詢中，使用者可以透過哼唱一段旋律或是由鍵盤輸入一段樂曲的方式進行查詢的動作(Query By Example)，然而使用者的記憶與現實的音樂片段，可能會有落差，並無法完全的利用精確比對(exact matching)達到效果。而可能需要允許使用者在查詢的過程中，包含多音、少音、升音與降音的可能發生，此時，允許容錯的近似搜尋(approximate searching)就相對的重要。而現有的近似音樂搜尋技術中[5][14][16]，多僅容錯查詢於小部份的錯音、多音或少音，對於升調、降調的部份則未受到討論或必須更進一步的以前後音差來比對，因此成效有限，仍然有很大的改善空

間。本研究的主要目的，即在於希望能提出一更有效率的音樂近似比對技術，使音樂近似搜尋更貼近於實際的應用。

2. 文獻探討

隨著多媒體資料在數量上的急速成長。如前所述，使用者在做查詢的動作時，往往與現實的音樂片段有所出入，允許容錯就成了一個重點，在這章節中，將對音樂與相關字串近似比對做進一步的說明。

2.1 音樂資料的相似性比對

Liu 等學者於[14]，提出了近似字串比對的音樂資料搜尋，主要是先萃取出音樂資料的相關特性，如旋律、節拍與合弦等，將其儲存為鏈結(link list)結構，成為音樂擷取的索引。再結合所提出的容錯演算法，在鏈結中做字串的容錯(fault tolerance)比對。此研究在容錯比對的方面，允許缺音、多音以及走音的比對搜尋。以下表 1 做一個例子說明:

表 1、容錯比對說明

字串	容許錯誤	可能結果
do-re-mi	缺音	do-re、re-mi、do-mi…
	多音	do-fa-re-mi、so-sol-re-mi…
	走音	do-re-?、?-re-mi、do-?-mi…

近似字串比對所建立鏈結索引的演算法，大致上可分為以下幾個步驟:

2.1.1 缺音連接(dropout errors)

又稱為允許越音的連接，將同一個音樂片段的連續位置以缺音連接(dropout link)。如圖 1，re(2;5)有連接到 mi(2;6)，而 re(2;5)卻沒從 start 做連接，則可以以缺音連接的方法從 start 連至 re(2;5)，而 re(2;11)也是同樣的情況以缺音連接，從 start 連至 re(2;11)，另外，re(2;10)前面有連接，亦可以缺音連接至 end 節點。

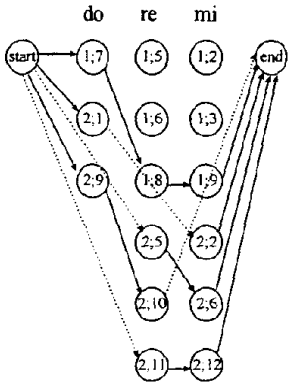

圖 1、缺音連接(dropout link)

2.1.2 多音錯誤(insert errors)

假設 re 的第五個節點(2;10)與 mi 的第六個節點(2;12)，可表示可能會有某個音發生在中間“re-?-mi”，則可以將其以多音的方式做鏈結(insertion link);而 do(2;9)也依此情形與 re(2;11)做鏈結，表示可能會有 “do-?-re”的相似音樂片段。如圖 2 所示:

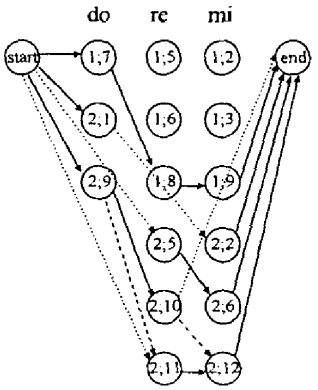

圖 2、多音連接(insertion link)

2.1.3 走音錯誤(transposition errors)

假設 re 第四個節點(2;5)並不是 M2 音樂旋片段的第一個音、而且與 mi 的第五個節點(2;6)做相連，若從 start 連接到 re(2;5)，則可能表示有某個音走掉了“?-re-mi”;依此類推，start 連接到 re(2;11)，而 re 節點又連接到 mi(2;12)，可以表示“?-re-mi”;而 do(2;9)有連接 re(2;10)，若在這直接連接到 end，則有走音的連接(transposition link)，可表為“do-re-?”的相近音樂片段。如圖 3 所示:

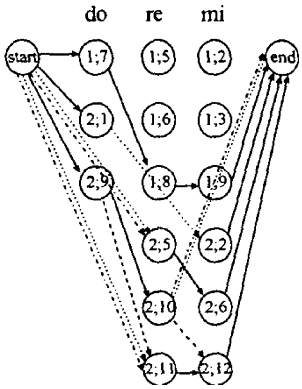

圖 3、走音連接(transposition link)

依以上步驟，可看出近似字串索引建立的情形，依照精確連接(exact link)、缺音連接(dropout link)、多音連接(insertion link)、走音連接(transposition link)之可能性，依序建立相關索引，但此演算法容錯能力僅限於一個音。

2.2 音樂資料的數值比對

Lo 等於 2002 與 2009 年提出了如何利用數值索引的近似搜尋技術[16][19]。首先，此技術是利用一音樂資料的轉換函數，將音樂旋律片段轉換成唯一的整數值，再將音樂之數值資料，建構於修改過的 R-tree 索引中，以方便進行查詢的比對。而做法大致可分為以下步驟:

(1)假設所能呈現音符(pitch)的種類為 n 個，例如:Do、Re、Mi…。而將每個音符階依序對應到 0、1、2、3 …的整數值。如果每次依序選取主音旋律連續的 m 個音符來建立索引，並且假設此連續 m 個音符為 P_1、P_2、…、P_m 音符所對應的數值可以以 P(x)來表示，$1 \le x \le m$ 即可利用音符的數值轉換函式 v(m)，將此連續 m 個音符轉換為唯一的整數值。

$$v(m) = \sum_{i=1}^{m} P(x_i) \times n^{i-1} \quad \ldots \text{公式(1)}$$

(2)接著，再利用所修改過的 R-tree 來建立索引，假如不同的音符種類有 10 個，符號表示為'*a*'、'*b*'、…、'*j*'，且所應到的數值為'0'、'1'… '9'。現有兩隻老虎(S1：Do Re Mi Do Do Re Mi Do)及小蜜蜂(S2：Sol Mi Mi Fa Re Re)兩首的音樂片段，此兩首所對映到的符號為"abcaabca"及"eccdbb"。又假設建立索引每次所選取的連續音符 m 為 4，則兩隻老虎的片段可拆成"abca"、"bcaa"、"caab"、"aabc"與"abca"五個片段。接著，利用數值轉換函式，將 "abca"轉換為數值的 210 其轉換過程如下 ($0\times10^0+1\times10^1+2\times10^2+0\times10^3$)，其餘的片段依序對應到 21、1002、2100、210 等五個數值。同理，小蜜蜂也是對應到 3224、1322、1132 等三個數值。利用 R-tree 所建立的結構，如圖 4 所示:

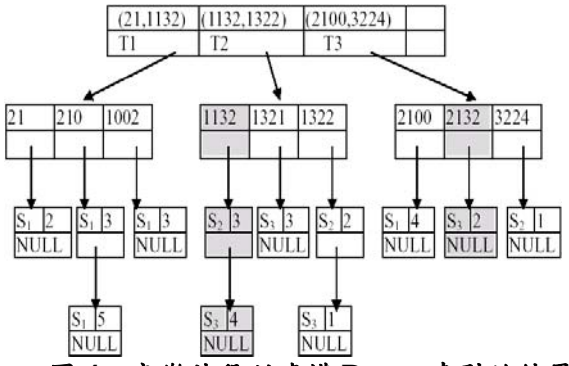

圖 4、音樂片段所建構 R-tree 索引的結果

(3) 接著，在此索引中做近似的比對，其方法有二:第一種搜尋方式是將是將所有可能的誤差結果都列出來，再將這些誤差結果全帶入索引做比對。第二種搜尋方式是利用數值的差值來做比對。首先，假設 k 為允許音符錯誤的個數，h 為允許的音差，帶進 R-tree 索引做比對，在比對前，一樣先將查詢值以及所有可能的音樂片段利用公式，轉換成對應的數值。

第一種搜尋方式:假設有一查詢值為 "cdbb"，如果我們允許一個錯音(k=1)，則將所有的可能值列出，分別為"cdbb"、"ddbb"、"bdbb"、"cebb"、"ccbb"、"cdcb"、"cdab"、"cdbc"、"cdba"，共有九種可能。利用公式，所對應到的數值又分別為 1132、1133、1131、1142、1122、1232、1032、2132 與 132。將所有的數值帶入 R-tree 中做比對，可分別找出吻合的 1132 與 2132 兩個數值，如圖 5，鏈結下方對應到 S_2 與 S_3，此兩個音樂片段就可能為查詢者所要找尋的音樂。

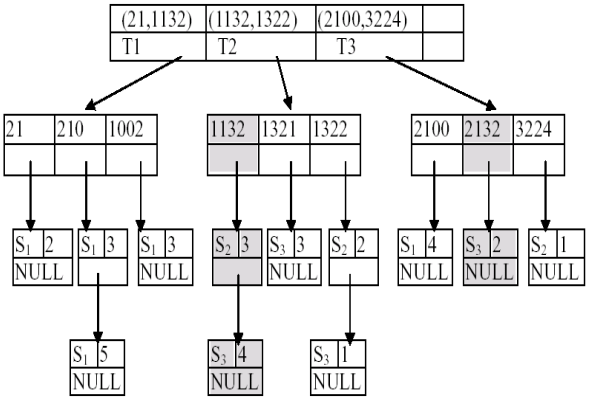

圖 5、採用近似比對所對應到 R-tree 的節點

第二種搜尋方式:利用數值的差值做比對。假設條件與第一種搜尋方式相同，允許 1 個音符的錯誤(k=1)，以及音高誤差的範圍為±1 度(h=1)。將要查詢的音樂片段轉換成數值，進入 R-tree 中做比對。舉例說明，如表 2:

假設在比對的過程中吻合第一列的條件，即葉節點與所轉換的值差的絕對值小或等於 1×10^0(=1)，則表誤差在第一個音，且只差了一度音，依此類推。

表 2、音樂數值差的絕對值可容許範圍(m=4)

(1)	<= 1×10^0
(2)	<= 1×10^1 ∧ a multiple of 10^1
(3)	<= 1×10^2 ∧ a multiple of 10^2
(4)	<= 1×10^3 ∧ a multiple of 10^3

在此 R-tree 中做比對的方法，允許容錯的能力演算法也只僅限於一個音。

2.3 近似字串比對技術

近似的字串比對(approximate string matching)，主要就是在一個字串 T(text)中找尋與給定的字串 P(pattern)，最近似的子字串(substring)。字串的比對又可分為循序性的字串比對(sequential string matching)[21]，及多重樣式的近似比對(multi-pattern approximate matching)[21]，主要研究又以循序性的字串比對為主。

使用者的音樂查詢，可能會因為記憶的模糊或只是大概的印象，而與現實的音樂片段有所出入，這時則可以透過近似的字串比對，在音樂資料庫中，比對出與使用者所輸入的查詢，最相似的音樂片段(字串)，相關公式如下:

(1) 李文斯頓距離 Levenshtein distance (edit distance)[9]

最常用來做為字串間的錯誤檢查，比較兩兩字串間的距離(distance)。所謂的 distance 是指字串 A 需要經過多少的 insert、delete、revise 才可以轉變成字串 B。當計算出來的 distance 越小則表示字串間的相似度越高。

假設有一字串 A 為 “tough”與字串 B 為 “dughe”，所對應到的 distance 如下圖 6。

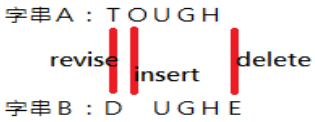

圖 6、李文斯頓距離(distance 為 3)

1. 插入(insert):表示字串 A 中擁有的字元而字串 B 中沒有。
2. 修改(revise):字串 A 字元與字串 B 字元的不同。
3. 刪除(delete):字串 B 中擁有的字元有而字串 A 中沒有。

(2) 餘弦相似度(cosine similarity)[1]

常被拿來衡量文件間距離的度量方法，主要是以兩個 d 維的向量角度差來度量該向量間的距離，其所得的數據介於 0~1 之間。當兩個向量角越接近時，所求得的餘弦相似度距離越接近 1;否則越接近 0。公式如下:

$$\text{Cosine (A, B)} = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \text{...公式(2)}$$

(3) Dice 相似度 (Dice similarity)[6]

偏重於集合的交集、聯集觀點，其所得的數據介於 0~1 之間。當兩個向量越接近時，所求得的 Dice 相似度距離越接近 1;否則越接近 0。公式如下:

$$\text{Dice (A, B)} = \frac{\sum_{i=1}^{n} 2(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2} + \sqrt{\sum_{i=1}^{n} B_i^2}} \text{...公式(3)}$$

(4) Jaccard 相似度 (Jaccard similarity)

與 Dice 相似度一樣，偏重於集合的交集、聯集觀點。當兩向量交集數愈高，其相似度愈高，其所得的數據介於 0~1 之間。當兩者越接近時，所求得的 Jaccard 相似度距離越接近 1;否則越接近 0。公式如下:

$$\text{Jaccard(A,B)} = \frac{\sum_{i=1}^{n} 2(A_i \times B_i)}{\sum_{i=1}^{n} A_i^2 + \sum_{i=1}^{n} B_i^2 - \sum_{i=1}^{n} (A_i x B_i)} \text{...公式(4)}$$

3.研究方法

雖然音樂特徵可以用字串形式來表示，以幫助查詢比對，但由於音樂的容錯查詢應該可以允許錯音、多音、缺音、升調與降調，並不完全相同於字串的近似比對。我們簡化查詢比對流程為使用者提供查詢樣本(Query By Example)，經過特徵的提取轉化為音樂字串，與音樂資料庫中的資料進行近似比對，最後產生最近似的候選清單，流程如圖 7 所示。

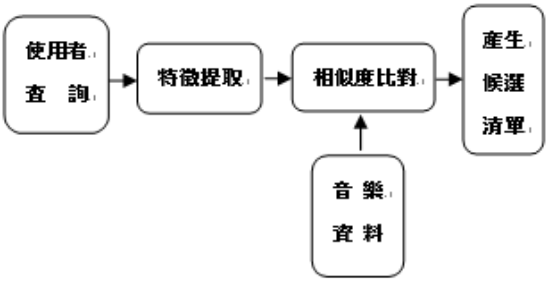

圖 7、音樂查詢流程圖

本研究主要目的在於相似度比對的改善，我們從現有的近似字串比對的公式中做分析，以找尋最適合於音樂字串比對的公式，期望再經過我們對公式的組合設計後，可以適用於近似音樂的查詢，讓音樂的近似查詢更為簡易而有效率。

為了能夠適當的評估音樂字串比對結果的近似程度，我們設計了音樂字串相似度的距離計算公式 $d(s, r)$如公式(5)。

$$d(s,r) = |s - r| \quad \text{...公式(5)}$$

其中 *s* 與 *r* 分別代表兩音樂旋律查詢樣本字串，但其相減是計算兩字串的差異程度，以多一個音、少一個音、或高低一度音，代表差距值為 1。兩字串的差距值越小，表示相似度越高，若差距值為 0，則為完全相等。計算方法說明如下：

(1) 多音—假設有“abcde”與“abde”兩音樂旋律字串，因“abcde”與“abde”相差一個音，則差距為 1。又如 abcde”比對於“abd”，多了兩個音差距為 2。

(2) 少音—反之假設有“abde”與“abcde”兩音旋律樂字串，因“abde”與“abcde”相差一個音，則差距為 1。又如“abd”比對於“abcde”，少了兩個音差距為 2。

(3) 錯音—假設有“abcde”與“abdde”兩音樂旋律字串，因“abcde”與“abdde”僅中間之 c 與 d 一度音之差別，則差距為 1。而如為“abcde”比對於“abfde”，其中 c 與 f 差兩度音，則差距為 2。

如此，“abcd”與“abdde”可視為其中 c 與 d 差一度音，且少一 e 音，兩音樂字串差距為 2。以此音樂字串近似比對方式，差距值越小的，應是最接近使用者所期望搜尋得之音樂標的。

由於給一音樂旋律的查詢樣本字串，可以分解為兩種的音樂特徵，如：「旋律(melody)」與「前後音差(tone difference)」(註：前後音差可不受整首音樂升降調-key 的影響)，若是將它們一一的以前述近似字串比對的公式於資料庫中做比對搜尋，則兩特徵所比對出來的最近似結果的音樂字串可能不同，其結果可能是由不同公式所找出的不同音樂字串。此外，除了從音樂旋律的查詢樣本字串中，可以很容易的取得旋律與前後音差的特徵外，我們也不排除使用者可能提供其它音樂特徵樣本來做為查詢，如：節拍(rhythm)、合弦(chord)等。但是，只要能將這些查詢樣本轉換為字串形式來表示，我們也希望能夠比對處理，而音樂資料轉換成字串的方式已於一些研究報告中被提及[12][13][14][17][18]，在此不多做討論。如果使用者提供越多種的音樂特徵同時來做查詢，往往也可以比對出更精確的結果。因此，在我們的研究中，除了將以不同音樂特徵個別用於前述近似字串比對的公式中做分析外，也將組合這些字串比對公式，以適用於分析音樂資料多特徵的情形，一起對特徵做比對，並據以尋求最佳的組合方式與可能的最佳結果。如此，適用於分析多特徵與多近似字串比對的最短距離公式將如公式(6)。

$$Min(s,r) = Min_{i=1}^{v}[Min_{j=1}^{k}\{d_{公式j}(s_i,r_i)\}] \ldots 公式(6)$$

公式中，*s* 與 *r* 分別代表兩個音樂資料，*v* 代表有多少種音樂特徵，*k* 代表有多少近似字串比對公式要計算，s_i 與 r_i 則分別代表 *s* 與 *r* 的第 *i* 個特徵。而 $d_{公式j}(s_i, r_i)$是代表以近似字串比對公式 *j* 與距離公式，來計算 s_i 與 r_i 兩音樂特徵字串的距離。接著，等號右邊第二個 *Min* 是指各音樂特徵經過 *k* 個近似字串比對公式計算，求其最小距離。等號右邊第一個 *Min* 指在 *v* 種音樂特徵中要分別去計算距離，並找出最小值。函式 *Min*(*s*, *r*)是指比較 *s* 與 *r* 的可能特徵，尋得最小距離。

4. 效能評估

於此章中，我們將第二節中所介紹的現有近似字串比對技術 Cosine similarity、Dice similarity、Jaccard similarity 與 Edit distance 等，用來做實驗分析，並擇優做兩兩組合，以了解它們應用於音樂字串比對之可行性。

4.1 實驗設定

由於使用者最容易提供的查詢樣本為音樂的旋律，且透過音樂旋律，我們也很容易計算出其旋律的前後音差，因此，我們的實驗將以旋律(melody)與前後音差(tone difference)兩特徵來進行。參照已發表之研究實驗方式[12][13][14][16][17][18][19]，在實驗中，我們收集了 200 首不同的音樂，分別轉換成音樂旋律字串後，再將這些音樂字串之字尾(suffix)[20]儲存於資料庫中，因此，總共儲存有一萬五千多筆之字尾資料，以供查詢比對使用。由於鋼琴鍵盤共有 88 個音，但在音樂主旋律中，多數的鍵是很少用的，因此，為了讓實驗單純，我們在實驗前先對所收集的 200 首音樂的旋律去做分析，以了解各個音(note)的出現頻率，結果如圖 8。結果發現多數的音發生於低音的 Mi~Si、中音的 Do~Si、高音的 Do~Si、以及高高音的 Do 等 20 個音，我們依序以‘a’、‘b’、 …、‘t’符號來代表這些音，並將其他出現頻率非常少的低音與高音皆分別歸屬於‘a’與‘t’，則我們將音符單純化為 20 個，此即為我們資料庫中所儲存的音樂資料形式。如此，我們實驗中兩個音的前後音差的可能範圍則為 -19, -18, …, 0, 1, …, 19 等，如：“ab”為 1、“ba”為-1、“at”為 19、“ta”為-19，共 39 種可能。因此，我們同樣可以用 39 符號來代表，使得音樂的前後音差也可以很容易的轉換為字串的形式來表示，以方便做近似字串的比對。

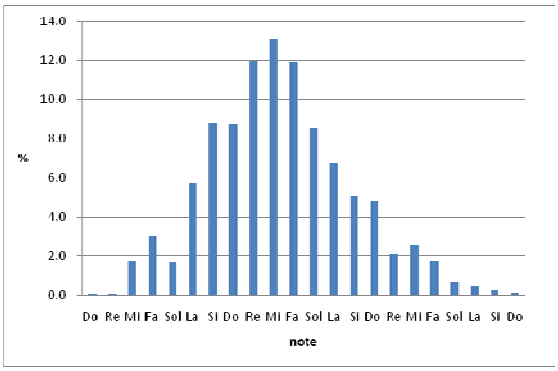

圖 8、旋律分佈圖

接著，我們實驗可分為如下 4 個步驟：

(1) 從資料庫中隨機取得長度 5~10 之音樂樣本字串。

(2) 將取出的音樂字串設計為多音(多 1 個音)、少音(少 1 個音)、錯音(錯 1 個音且在±3 度音間)、以及升調(全升高一度音)等，以做為查詢樣本。我們也特別註明，雖然實驗所採用的近似字串比對技術並不局限於此，也能比對多或少多個音或錯多個音的情況，但可能會造成有多個查詢結果，而使實驗分析複雜化，因此，如本節開始所述，參酌現有近似音樂比對的研究，我們也採單純化的實驗，以方便做分析。

(3) 分別對前面介紹的四種近似字串比對公式做不同特徵樣本的實驗，並以我們所設計的公式(5)來計算查詢結果與查詢樣本間的差異值，於資料庫中比對出各方法所找到之最近似結果。

(4) 除了對音樂旋律與前後音差做單特徵的分析外，我們也做了組合近似字串比對公式的實驗，並以公式(6)來判斷比對結果。

每一實驗做超過一百次，並計算查尋結果與查詢樣本的差距平均值。實驗的分析部份分為二大主題，分別為:

(1) 單特徵的實驗分析

(2) 多特徵的合併實驗分析

4.2 單特徵的實驗分析

此實驗，我們主要是針對四個不同的公式做單特徵的實驗分析，在特徵值的部份，我們分別取用了旋律及前後音差等二個音樂特徵來做實驗，實驗結果分別呈現於圖 9~圖 16 中。以下我們分別對每個實驗結果做分析與討論。

4.2.1 單特徵-旋律

在這節裡，將針對音樂的旋律特徵，以查詢樣本比實際資料庫中的資料多 1 個音、少 1 個音、錯 1 個音、以及升調 1 度音，來做實驗。

4.2.1.1 查詢樣本-多音

此部份的查詢樣本，主要是分別從音樂資料庫中，以亂數方式分別為長度 5~10 的樣本各取出 100 筆的原始音樂資料，若所取出之資料太長，則僅採用其前面幾個音，再將每一筆音樂資料任意的插入一個音，以做為多音的查詢樣本，在經由四個近似字串公式的比對，分別找出資料庫中與所要查詢的音樂字串距離值最小的為結果，並各別計算每一公式經 100 個樣本測試後的平均差距值，實驗的查詢結果如圖 9 所示。於圖 9，我們可以得知除了 Edit distance 皆保持在差距值為 1 的實驗結果，其他的三個公式，都會隨著查詢樣本長度的增加而差距越來越大。也就是以 Edit distance 公式來做多音的比對，似乎比其他公式來得準確。

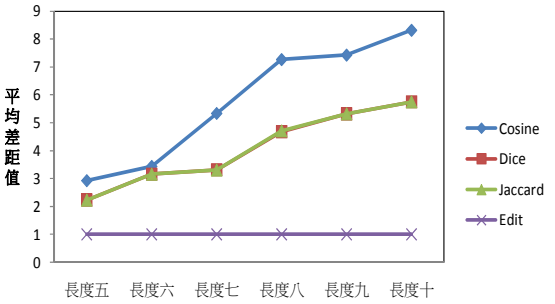

圖 9、旋律特徵-多音之平均差距

4.2.1.2 查詢樣本-少音

此部份的查詢樣本取得方式與前一節一樣，只差每一筆由音樂資料庫中取得的原始音樂旋律字串，我們刪除了其中任意的一個音，以做為少音的查詢樣本，也同樣有 100 個樣本做實驗，經由實驗的查詢結果，如圖 10 所示，我們可以得知除了 Cosine similarity 的結果起伏較大，很不穩定外，其他的三個公式，皆能保持在平均差距值於 1 以下，而 Jaccard similarity 與 Edit distance 則有較佳的實驗結果。

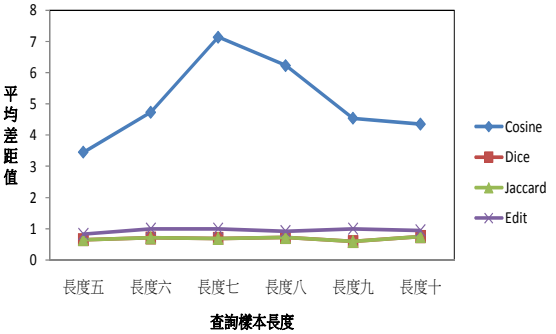

圖 10、旋律特徵-少音之平均差距

4.2.1.3 查詢樣本-錯音

此部份的查詢樣本取得方式如前所述，但每一筆由音樂資料庫中取得的原始音樂旋律字串，我將其中的一個音以任意的方式做-3 到+3 的音階改變，以產生錯音的查詢樣本，經由實驗的查詢結果，如圖 11 所示，我們可以得知除了仍然以 Cosine similarity 的結果起伏較大，效果較差外，其他的三個公式較優，這次則是 Dice similarity 及 Jaccard similarity 表現略優於 Edit distance。

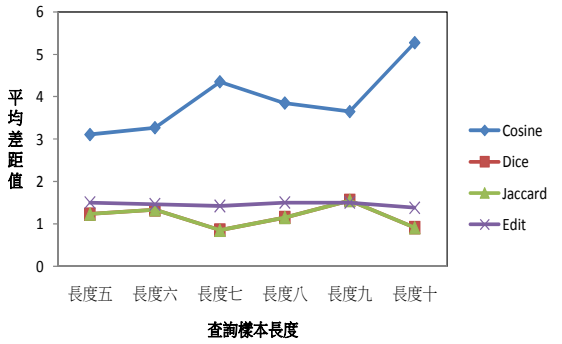

圖 11、旋律特徵-錯音平均差距

4.2.1.4 查詢樣本-升調

使用者做音樂查詢時，經常可以用哼唱的提供查詢樣本，除了可能多音、少音、錯音外，也經常會升調或降調，這情形並不代表樣本有錯誤，前後音差則可不受此升降調的影響，因此成為一很重要的音樂比對依據。也因升調或降調對前後前後音差不影響，我們只取升高一度音來做實驗。此部份的查詢樣本取得方式仍然如前所述，但每一筆由音樂資料庫中取得的原始音樂旋律字串，我們將其整個升高 1 度音，以表示用者升了一個 key，以產生升調的查詢樣本，經由實驗的查詢結果，如圖 12 所示，我們可以得知，Cosine similarity 的結果仍然隨著查詢樣本長度的增加而產生較大的起伏，而 Dice similarity 及 Jaccard similarity 的實驗結果不但相似， 也是實驗圖中較佳的兩個結果。

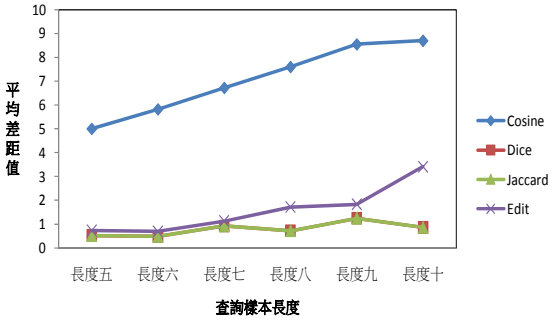

圖 12、旋律特徵-升調之平均差距

4.2.2 單特徵-前後音差

在這節裡，我們做音樂前後音差的實驗分析。首先我們如旋律的分析一樣，從資料庫中任意取得的原始音樂旋律字串做多 1 個音、少 1 個音、錯 1 個音、以及樣本整個升 1 度音的處理，在將這些音樂字串轉換為前後音差的字串，以做為查詢樣本進行實驗。也同樣對每個近似字串比對公式，每次也都是取 100 個樣本做實驗，以計算平均距離值。

4.2.2.1 查詢樣本-多音

前後音差的多音實驗，其查詢實驗結果如圖 13 所示，由圖中我們可以得知，實驗的四個不同的近似字串公式，皆會隨著音樂片段長度的增加，而增加其差異性。其中 Dice similarity 與 Jaccard similarity 的結果也極為相似，是此實驗中表現最佳的。

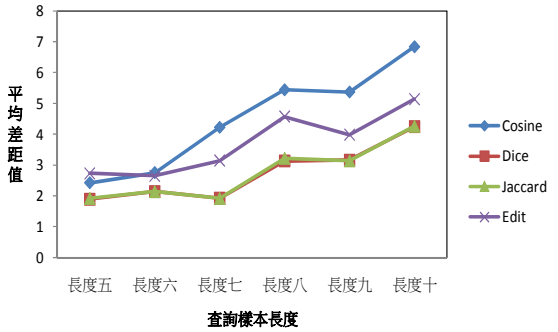

圖 13、音差特徵-多音之平均差距

4.2.2.2 查詢樣本-少音

前後音差的少音實驗，其查詢實驗結果如圖 14 所示，如圖可以得知除了 Cosine similarity 的結果起伏較大以外，其他的三個公式，皆能保持在平均差距值於 1 以下，Dice similarity 與 Jaccard similarity 的結果仍然極為相似，相較於其他三個公式，Edit distance 為此實驗圖最佳的結果。

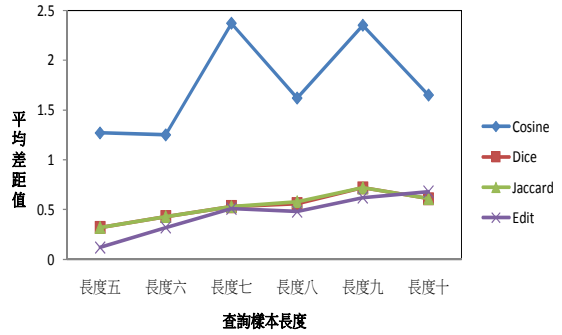

圖 14、音差特徵-少音之平均差距

4.2.2.3 查詢樣本-錯音

前後音差的錯音實驗，其查詢實驗結果如圖 15 所示，圖中除了 Cosine similarity 的結果起伏較大以外，Edit distance、Jaccard similarity 及 Dice similarity 在長度五到長度六的部份有一部份的交錯，若查詢的音樂樣本較長，Jaccard similarity 與 Dice similarity 為此實驗中較佳的結果，而兩公式表現幾乎相同。

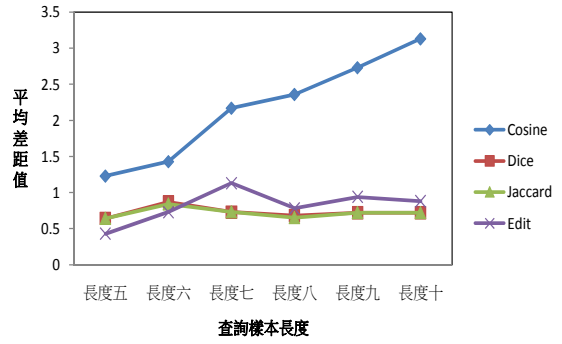

圖 15、音差特徵-錯音之平均差距

4.2.2.4 查詢樣本-升調

前後音差的升調，其查詢實驗結果如圖 16 所示，我們可以得知 Edit distance 皆能在不同的長度的查詢中找出最佳的查詢結果，其原因就在於前後音差不受升調或降調的影響，可以保持欲查詢目標的前後音差不變，以 Edit distance 可以精確的找到目標。反觀其他三個公式，則都產生不規則的改變結果，其結果相較於 Edit distance 則遜色許多。

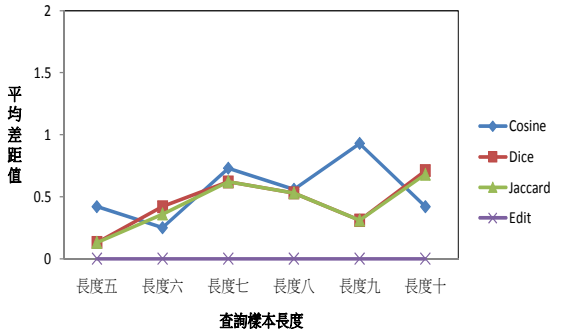

圖 16、音差特徵-升調之平均差距

4.3 多特徵的合併實驗分析

在以單特徵的比對分析後，我們發現不同的近似字串比對公式分別對旋律與前後音差各有偏好。而若音樂資料的查詢比對若可以以多特徵的方式來進行，則可以增加比對結果的精確度。在本節的實驗中，我們將近似字串比對公式做兩兩的搭配，並以公式(2)進行最小差距值的判斷，以進行多特徵的研究實驗。由於我們在單特徵的實驗中可以發現 Dice similarity 與 Jaccard similarity 二個近似字串比對公式，在每個實驗圖中的表現都極為相似，為了簡化實驗與節省報告的篇幅，在此節的組合實驗，我們將僅以 Jaccard similarity 為代表，只利用 Cosine similarity、Jaccard similarity 及 Edit distance 三個公式，來做組合實驗，因此我們會有(Cosine similarity + Jaccard similarity)、(Jaccard similarity + Edit distance)、(Cosine similarity + Edit distance)、(Cosine similarity + Jaccard similarity + Edit distance)等四種組合情形。而我們接下來多音、少音、錯音與升調的分析，皆代表兩特徵—“旋律”與“前後音差”的實驗。其餘如查詢樣本的設定與取樣數量，也皆與單特徵實驗的情形一樣。

4.3.1 查詢樣本-多音

此節進行多特徵—多音的實驗，實驗結果如圖 17 所示，我們可以看到(Cosine similarity + Edit distance)、(Jaccard similarity + Edit distance)與(Cosine similarity + Jaccard similarity + Edit distance)三個組合在不同的查詢長度下的表現皆是相同的，一直保持在相對低的平均差距值。若對照第 4.2.1.1 節單特徵旋律—多音的實驗結果與第 4.2.2.1 節單特徵前後音差—多音的實驗結果，應該是 Edit distance 主導了實驗結果。而(Cosine similarity + Jaccard similarity)的組合，則會隨著查詢長度的增加跟隨著增加其差異性。

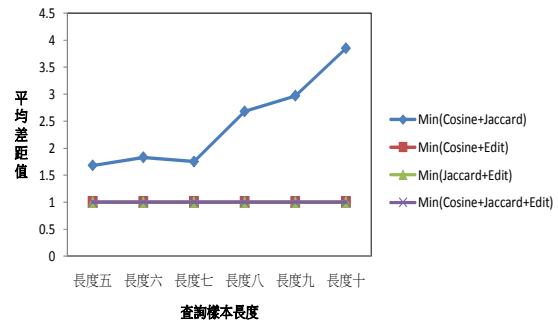

圖 17、多特徵-多音之平均差距

4.3.2 查詢樣本-少音

此節進行多特徵—少音的實驗，實驗結果如圖 18 所示，我們可以看到三個不同的組合皆會隨著查詢長度的增加跟隨著些微的增加其平均差距值，而其中除了(Cosine similarity + Jaccard similarity)於音樂樣本小時略遜色外，其他實驗結果四組合皆很接近。

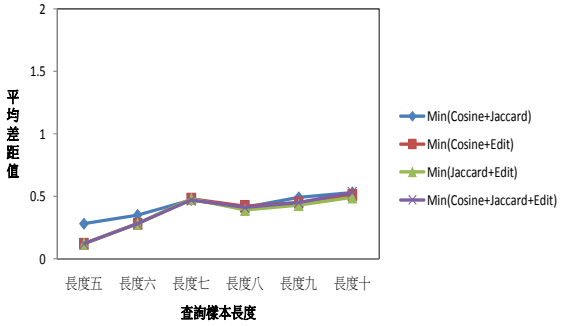

圖 18、多特徵-少音之平均差距

4.3.3 查詢樣本-錯音

此節進行多特徵—錯音的實驗，實驗結果如圖 19 所示。圖中四組合的平均差距值，仍然是隨查詢樣本長度的增加，有往上走之趨勢。而其中，則是(Jaccard similarity + Edit distance)與(Cosine similarity + Jaccard similarity + Edit distance)的兩組合，曲線接近，也有較佳的實驗結果。

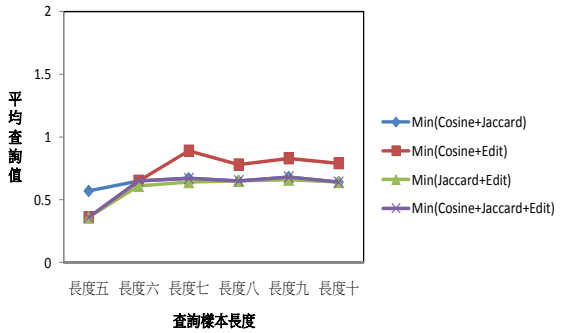

圖 19、多特徵-錯音之平均差距

4.3.4 查詢樣本-升調

最後我們進行多特徵—升調的實驗，實驗結果如圖 20 所示，我們可以了解到(Jaccard similarity + Edit distance)、(Cosine similarity + Edit distance)與(Cosine similarity + Jaccard similarity + Edit distance)表現較佳，且相當。應該仍然得利於前後音差不受升調的影響，Edit distance 可以很精確的比對出欲查詢的結果。

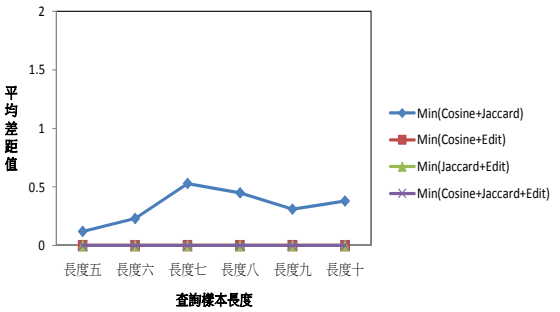

圖 20、多特徵-升調之平均差距

5.結論

音樂資料的樣本查詢，往往無法保證使用者的音感是準確的，往往會有些許的差異產生，如多音、少音、錯音與升降調等等，因此，允許容錯(fault tolerance)的機制就相對的重要。音樂內容的擷取技術，多將音樂特徵轉換成音樂字串以供查詢比對，但是音樂字串的近似比對，又與一般字串比對不完全相同，我們的研究，設計了多特徵比對的公式，接著從現有的近似字串比對的技術中，進行了實驗分析，以了解它們應用於音樂字串比對的可行性，而從實驗結果我們發現，因為每個公式在字串比對上皆有其特性，我們只要有效的將不同的公式做合併，並結合於不同容錯情況，其中(Cosine similarity + Jaccard similarity + Edit distance)表現好是必然的，但是必須做更多的計算，而(Jaccard similarity + Edit distance)有相似效果，應該就足以用來做音樂旋律與前後音差兩特徵的近似比對。。

未來我們將繼續找尋各種可能應用於音樂字串查詢的相關公式，並分析現有字串近似的比對技術應用於音樂字串比對之優劣點，考慮融合各現有技術的優點並加以改良，或是自行開發更適用於音樂字串查詢的公式。此外，我們將持續擴大音樂資料庫的大小以及擴大容錯條件與做更完整之樣本測試，使之更適用於音樂資料之近似比對，也使音樂查詢的處理能更有效率。

參考文獻

[1] R.J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all-pairs similarity search," *WWW Conference,* pp. 131-140, 2007.

[2] 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.

[3] James C.C. Chen and Arbee L.P. Chen, "Query by Rhythm An Approach for Song Retrieval in Music Databases," *In Proc. Of Int'l Workshop on Research Issues in Data Engineering*, pp. 139-146, 1998.

[4] M.T. Chen and J. Seiferas, "Efficient and Elegant Subword-Tree Construction," *Springer-Verlag, New York*, 1985.

[5] B. Cui, L. Liu, C. Pu, J. Shen, and K. Tan, "QueST: Querying Music Databases by coustic and Textual Features," *Proceedings of the 15th international conference on Multimedia*, pp. 1055-1064, 2007.

[6] L.R. Dice, "Measures of the Amount of Ecologic Association Between Species," *Ecology 26*, pp. 297–302, 1945.

[7] E.A. El-Kwae and M.R. Kabuka, "Efficient Content-Based Indexing of Large Image Databases," *ACM Trans. On Information Systems* (18:2) 2000, pp. 171-210.

[8] S.T. Goh and K.L. Tan, "MOSAIC: A Fast Multi-Feature Image Retrieval System," *Data and Knowledge Engineering* (33:3) 2000, pp. 219-239.

[9] P. Hall and G. Dowling, "Approximate String Matching," *ACM Computing Survey*, Vol. 12, NO.4, pp. 381-402, Dec. 1980.

[10] K.A. Hua, K. Vu, and J.H. Oh, "SamMatch: A Flexible and Efficient Sampling-Based Image Retrieval Technique for Large Image Databases," *Proc. of Multimedia '99 Florida,* Oct., 1999, pp. 225-234.

[11] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura." *Bulletin de la Société Vaudoise des Sciences Naturelles 37*, pp. 547-579, 1901.

[12] W. Lee and A.L.P. Chen, "Efficient Multi-Feature Index Structures for Music Data Retrieval," *In Proc. Of SPIE Conf. on Storage and Retrieval for Image and Video Database*, 2000.

[13] C. H. Lin and Arbee L. P. Chen, "Indexing and Matching Multiple-Attribute Strings for Efficient Multimedia Query Processing," *IEEE Transactions On Multimedia*, Vol. 8, No. 2, April 2006.

[14] 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*, pp. 451-456, 1999.

[15] C.C. Liu and Arbee L.P. Chen, "3D-List: A Data Structure for Efficient Video Query Processing," *IEEE Transactions on Knowledge and Data Engineering* (14:1) 2002, pp: 106-122.

[16] Y-L. Lo and Shiou-jiuan Chen, "The Numeric Indexing For Music Data," *Proc. of the 4th International Workshop on Multimedia Network Systems and Applications (MNSA'2002)*, Vienna, Austria, pp. 258-263, 2002.

[17] Y-L. Lo, W-L. Lee, and L-H. Chang, "True Suffix Tree Approach for Discovering Non-trivial Repeating Patterns in a Music Object," *Journal of Multimedia Tools and Applications*, Springer, Vol. 37, No. 2, April 2008, pp. 169-187.

[18] Y-L. Lo, C-H. Lee, and C-H. Wang, "Scalable Multi-feature Index Structure for Music Databases," *Information Sciences*, Elsevier, Vol.179, Issue 15, July 2009, pp. 2662-2675.

[19] Y-L. Lo and Ling-Yi Tsai, "Approximate Searching for Music Data in Real-Valued Feature Indexing," *Journal of Convergence Information Technology*, vol. 4, no. 4, December 2009, pp. 87-95.

[20] E. McCreight, "A Space-Economical Suffix Tree Construction Alogorithm," *Journal of the Association for Computing Machinery* (23:2) 1976, pp:262-272.

[21] G. Navarro, "A Guided Tour to Approximate String Matching," *ACM Computing Survey*, Vol.33, No. 1, pp.31-88, Mar.2001.

[22]. S. Nepal and M.V. Ramakrishna, "Query Processing Issues in Image (Multimedia) Databases," *Proce. of Data Engineering,* Washington, 1999, pp. 22-29.

[23] T. Phillips, "Approximate string matching," *C/C++ Users J.*, Vol. 12, No.4, Apr. 1994.

[24] S.W. Smoliar and H.J. Zhang, "Content-Based Video Indexing and Retrieval," *IEEE MultiMedia* (1:2) 1994, pp. 62-72

[25] E. Ukkonen, "On-Line Construction of Suffix Tree," *Algorithmica* (14:3) 1995, pp:249-260.