國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

以雙向檢索及排序學習演算法來改進音訊指紋辨識

Improving Audio Fingerprinting by Bi-directional Retrieval and Learning to Rank

唐子翔

Tzu-Hsiang Tang

指導教授：張智星 教授

Advisor: Jyh-Shing Roger Jang, Ph.D.

中華民國一百零九年一月

January, 2020

doi:10.6342/NTU202000440

誌謝

三年的碩士生涯說短不短，一邊工作一邊求學蠟燭兩頭燒的辛苦讓畢業的果實嚐起來特別甘甜。我要萬分感我的指導老師：張智星教授的大力幫助，才能讓這份論文順利誕生。老師很大方地接納了半路出家的我，讓我一起參與 MIR Lab 裡核心的研究課題：語音辨識，更花了不少時間指導我，甚至配合我的上班時間，晚上播空線上指導我的研究，讓我得以順利畢業，感激之情言語難以表述萬一，很榮幸也很慶幸能遇到老師成為 MIR Lab 的一分子。

接下來要感謝我的太太在這三年時間裡的支持與包容，讓我可以奢侈地花費三年時間完成我的求學目標，毫無後顧之憂，是我最堅實的後盾。

最後要感謝實驗室的葉子學長、翔宇、濬慶，在我不熟悉實驗室運作方式而遇到問題的時候，三位分別多次幫助我解決問題，葉子學長在研究上的建議與指教更是惠我良多。感謝以上所有人的幫助與成全，不勝感激。

doi:10.6342/NTU202000440

摘要

音訊指紋辨識是一種快速且成熟的音樂檢索手段，使用者輸入其藉由麥克風錄製的一段音訊，讓系統抽取該音訊片段的特徵，再與資料庫中的歌曲特徵進行比對，最後輸出符合程度最高的結果給使用者。本篇論文將以在噪音環境下隨機錄製的音訊當作查詢片段，模擬在現實生活中錄製的音訊的過程與結果，嘗試找出針對抽取歌曲特徵這一步驟的改良方法。

在一個以地標為特徵基礎的音訊指紋系統中，我們嘗試改變組成地標方式與內容，也進一步改良地標的檢索方式。其中雜湊表中包含的訊息越多，就可以用更多的條件進行過濾，需要比對的地標數量也會隨之減少，輸出配對成果的速度也會隨之提高。我們也改進了檢索地標的方式，藉由雙向檢索 (bi-directional retrieve) 得到更多對辨識結果有正向幫助的資訊，來將初始的配對結果進行二次評分，初步提高辨識結果的準確率，接著利用排序學習演算法 (learning to rank) 來重新排序評分結果，使得辨識率進一步提高。

關鍵字: 音樂檢索、音訊指紋系統、地標、雙向檢索、排序學習演算法、AdaRank

doi:10.6342/NTU202000440

Abstract

Audio Fingerprint (AFP) Recognition is well known as a rapid and mature strategy in audio information retrieval. End user records an audio snippet as the input of our AFP system, the system would extract the features of the input snippet, then it would compare the features of snippet with the features in database which is formed by selected audio data set (known as ground truth). Finally the system returns the most likely match with details (song name, author, ..., etc) from database. In this thesis, we would randomly record voices with highly noise-affected environment, and set these audio snippets as query piece (input). We use these query piece to simulate the audio record in real life, and try to find a method to improve the way we used for feature extraction.

Based on the AFP system which uses landmark as basic feature, we try to change the content in the landmark to format different kind of Hash table. The more information contained in a Hash table, the more criteria we can use to filter the landmarks, and then we can check fewer landmark to get match result, this reduces the query time. We also improve the method for retrieving the hash table. Via Bi-directional Retrieve, we can get much more positive information from the same hash table to re-rank the match result, and increase the accuracy of match result. Further more, we use the algorithm from learning to rank to re-rank the match result, and then get the better accuracy.

Keyword : audio retrieval, audio fingerprint system, landmark, bi-directional retrieval, learning to rank, AdaRank

doi:10.6342/NTU202000440

目錄

doi:10.6342/NTU202000440

doi:10.6342/NTU202000440

圖目錄

doi:10.6342/NTU202000440

doi:10.6342/NTU202000440

表目錄

doi:10.6342/NTU202000440

第一章　導論

1.1 研究背景

自人類文明發現大自然的旋律並定義了音樂以來，音樂貫穿了整個人類文明。普遍認為人類始祖在分散在世界各地分化為不同人種之前已知曉音樂。目前推論音樂很可能已存在 55,000 年以上 [4] 。音樂不但在藝術文化裡扮演重要角色，休閒娛樂中它也一直都是主流活動之一。近 20 年，隨著多媒體設備的快速發展，加上行動網路技術提升帶來的便利性，從廣播收音機、CD 隨身聽、MP3 player 到現在的智慧型手機／平板，音樂對我們來說觸手可及。這也使得我們時常會有一種經驗：當我們在外面活動時，也許是路旁商家播放，又或者是路人手機傳出來一段旋律，讓我們覺得這首歌真好聽，想要收藏卻又不知道歌名，甚至可能連歌手是誰都不清楚，也缺乏線索可以查詢歌曲的相關資訊。這時候我們就需要一個方便且可靠的檢索系統 (Retrieval System) 幫助我們快速地瀏覽查詢，並找到其中有用的資訊。現代生活中已有許多文字檢索系統與演算法被開發出來並應用於產業中，如 Google Search、Microsoft Bing Search、Yahoo Search 等。而專門用於音樂的檢索系統也所在多有，其中藉由音訊指紋 (Audio Fingerprinting, AFP) 所開發出來的應用也有不少成熟並商業化的產品，其中有著許多知名的服務，如 SoundHound [5]、Shazam [6]、Echonest [7] 等等，我們藉由手機 APP 錄製歌曲中的一段聲音，透過網路連線將音訊傳至伺服器進行查詢比對，回傳最有可能的歌曲結果給使用者。近年來更由於智慧型手機的崛起與使用者需要在移動裝置上取得資訊的強烈需求，促使更多使用者注意到這類型的應用，它們依靠過去累積的龐大資料庫與快速檢索的技術滿足龐大使用者的需求，市場前景可期。

本論文所探討的音樂資訊檢索，是基於 QBEE (Query-By-Extract-Example) 發展而來的一種方法- AFP (Audio Fingerprinting)，它的應用除了上述的音樂資訊檢

doi:10.6342/NTU202000440

索以外，也常常利用其善於偵測與原曲相似度的特性來判斷是否有違法的侵權行為，微軟也曾經研究過 AFP 並嘗試建構相對應的偵測系統來偵測是否有複製行為[8]。也因此系統對使用者輸入的查詢片段有比較高的要求，查詢片段的特徵必須與資料庫內的原曲高度相似。在這樣的使用情境下，因為設備或者環境所造成的噪音便成為影響最大的干擾因子。有鑑於此，如何對抗干擾、減少儲存資料庫所使用的空間大小、減少辨識查詢片段所花費的時間以及提高回傳歌曲資訊的正確率，這些課題便成為相關研究領域的主要討論方向。

1.2 研究方向

本論文之研究方向為：使用以地標為辨識特徵的音訊指紋系統，針對組成檢索特徵的方法，在可接受的資料庫大小以及辨識時間下，提升系統辨識結果的正確率。主要內容包含以下幾點：

- 本論文是以 A. L. Wang 所提出的方法為基礎 [1] 來建構音訊指紋系統，Wang 使用地標作為音訊指紋，其是一系列基於音訊內容的訊號－包括頻域與時域，總結了該音訊的性質。我們將已知歌曲與其對應的地標配對並建立資料庫，而後將查詢音訊片段的地標拿來進行配對，最後找出資料庫中配對程度最高的歌曲，並將這首歌曲的相關資訊輸出給使用者。
- 接下來，本論文基於前人的研究結果 [3]，近一步改善使用 GPU 幫助運算所造成的正確率下降。使用 GPU 進行平行運算，可以有效降低在辨識時配對每一筆音訊指紋所花費的時間，若能將正確率提升到與單純使用 CPU 的系統相同，則可以有效提高系統的配對效率，增加系統面對大量檢索要求湧入時的抗壓能力。
- 第三步，本論文將針對組成地標的內容、地標比對的方法以及最後評分的方

doi:10.6342/NTU202000440

式進行改良，由於地標中包含了音訊的內容與性質，因此地標所包含的有效特徵越多，配對的準確率也會相應提升。但是在增加了地標所包含的特徵的同時，資料庫的大小也會隨之增加，每個地標需要進行的配對次數變多，這會大大影響辨識時所花費的時間長短，因此我們必須利用不同特徵之間的關聯性來進行過濾，進而降低比對的次數使得辨識時間變短，並利用不同的比對與評分方式來提高辨識結果的正確率，之後更進一步使用排序學習演算法重新排序配對結果，進而提升正確率。

- 最後，本論文將在資料庫大小、辨識所花費的時間、辨識結果的正確率三者之中取得平衡，以固定的資料庫大小為前提，將辨識所花費的時間控制在一個固定區間，並提升辨識結果的正確率。

1.3 章節概要

本論文之章節安排如下：

- 第二章：介紹本論文相關背景知識，包括 AFP 的系統與流程架構以及不同的 AFP 實作方法
- 第三章：介紹本論文所採用方法的各個步驟以及相關技術
- 第四章：介紹本論文對現有 AFP 系統的改良方法以及實驗結果的比較及分析
- 第五章：介紹本論文對於實驗結果的總結，並列出可能的改進方向與未來展望

doi:10.6342/NTU202000440

第二章　音訊指紋的相關研究

2.1 Wang's Method

此方法於 2003 年由 Avery L. Wang 所提出 [1]。該篇所提出的方法可以分成以下三個步驟：建立頻域－時域群集圖、抽取特徵建立地標並建構雜湊表、比對查詢片段與資料庫的地標並統計評分。而在產業界中，知名音樂搜尋引擎 Shazam [6] 便是運用此方法為基礎，進而衍伸而出的應用。Wang's Method 的優點包括可以在龐大的資料庫中利用雜湊表的特性快速地找到對應的地標，對於環境所造成的噪音與錄音器材造成的失真有極高的抗性，正確率上的表現也相當不俗。

2.1.1 建立頻域－時域群集圖 (constellations map)

在 Wang's Method 中，作者針對環境噪音以及訊號因 GSM 編碼壓縮而失真這兩個條件做了一連串的實驗，發現在時域－頻域群集圖中，若有一點相對於周邊訊號具有更高能量，作者將此點視作此音訊的峰點 (peak)，則峰點可以在充滿噪音以及因壓縮而失真的音訊片段中被保留下來。接下來，根據峰點的能量高低加上峰點分布的密度來選取突出點 (salient peak)，峰點密度越高的區域，則能量門檻值便會隨之提高，被選出的突出點其數量就越少，這樣一來就可以保證一段音訊的突出點可以均勻分布，充分保留該歌曲的特徵而不是偏重於某一片段。如此一來無論歌曲中的哪一段被擷取下來當作查詢片段，都可以在原曲中找到符合的突出點與之對應，進而提高系統的辨識能力。圖 2.1.A 顯示一段音訊經過傅立葉轉換後所形成的頻域－時域圖，灰階的濃淡表達該點能量的高低，越黑越高。圖 2.1.B 則顯示圖 2.1.A 經過取樣後，由突出點所組成的頻域－時域群集圖。此時整段音訊都已經過濾波器篩過一遍，各點能量都有所降低。若此段訊號是在高度干

doi:10.6342/NTU202000440

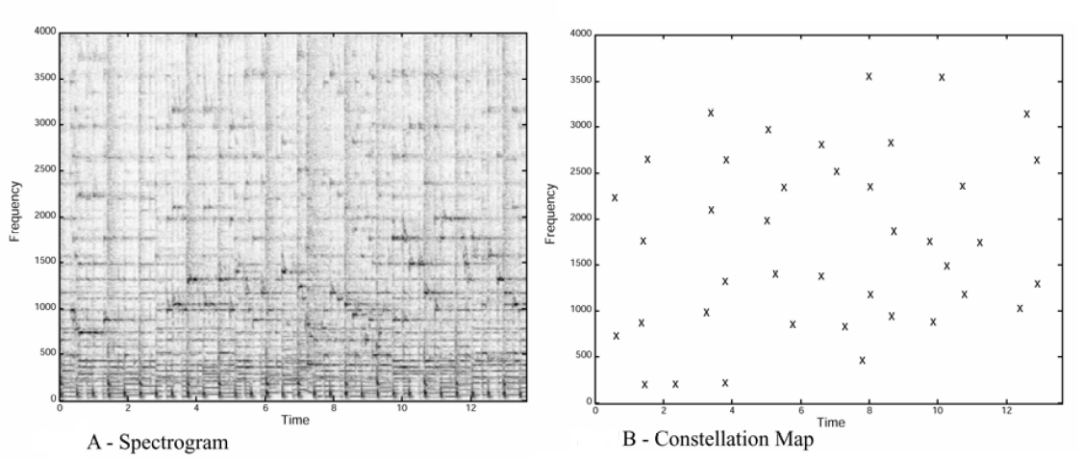

圖 2.1: 以頻譜為基礎建構頻域－時域群集圖 [1]

擾的環境下所錄製的片段，並將最後行成的群集圖拿去與原曲的群集圖作比較，其突出點的分佈仍會有高度的一致性。因為峰點是能量相對高的訊號點，加上突出點的取樣門檻值會隨著峰點密度及能量動態調整，因此兩張群集圖會有很大一部分的突出點是重疊的，而沒有重疊產生差異的部分，大多是因為少量高能量的噪音訊號出現導致該區域在取樣突出點時失誤，取出了未包含歌曲特徵的峰點。

2.1.2 根據地標建構雜湊表

接著我們討論如何在頻域－時域群集圖上，找出查詢片段跟原曲重疊的部分，若是要一個突出點一個突出點進行比對，再找出分佈重疊的片段，這樣的做法非常花費計算時間而且實作系統的設計也會相當複雜，因此該篇論文作者提出了一個效率與準確率兼顧的方法。首先在頻域－時域群集圖上對原曲及查詢片段進行索引：將頻域－時域群集圖上所有的突出點以特定方式形成“指紋”，再以歌曲的相關資訊 (曲名、ID 等等) 作為內容，兩者互相對應便可建構一張雜湊表，用來展現一首歌的內容 (時間點、頻率、能量等) 與標籤 (曲名、作者等)。建構“指紋”的具體為：先選取一個突出點點當做錨點 (anchor point)，當其被選取後，我們會在其後方按照事先選好的頻率差以及時間差定義一個屬於該點的目標區域

doi:10.6342/NTU202000440

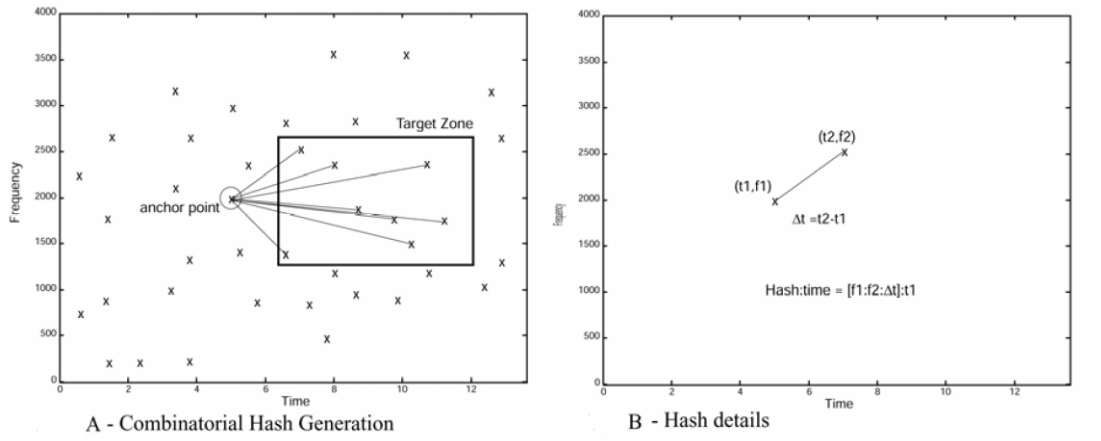

圖 2.2: 用峰點組成地標之示意圖 [1]

(target zone)。而後每個錨點都按照能量高低的順序，與其目標區域內部所有的峰點進行一對一配對，如圖 2.2.A 所示。我們接著計算每組配對 (連線) 中兩點的頻率差值以及時間差值，進而定義並組成一個恆為正值的雜湊鍵 (hash key)，在該篇論文作者也將此”指紋“稱為”地標 (landmark) “，如圖 2.2.B 所示。

接下來是建立雜湊表中每個鍵值對應的雜湊值，根據前述的方法，我們利用了歌曲的頻率及時間特性組成一連串的雜湊鍵，這樣不同歌曲之間，甚至是同一首歌曲的不同片段之間，都有可能找出相同的雜湊鍵。接下來我們必須要把歌曲的標籤放入雜湊值與雜湊鍵進行連結，藉由這張雜湊表，我們從查詢片段中取出的每一組地標，都可以在表中找到該查詢片段可能會對應到哪幾首歌曲。依循這樣的設計，我們將歌曲編號放入雜湊值，並輔以該雜湊鍵取樣時錨點的時間，兩者結合成完整的值，而對應到相同雜湊鍵但不同的值組合成一個集合，這個集合便是雜湊值。根據作者定義，雜湊鍵為 32-bit 大小的二進位 unsigned integer，雜湊值中每個元素的大小與雜湊鍵相同。定義完雜湊鍵與雜湊值的空間大小後，作者開始探討儲存資料庫的空間大小。這裡作者定義：每個錨點與目標區域之中若行成 F 個配對，則我們稱其展開大小 (fan-out 或 max pair per peak) 為 F；假設每個突出點都當作錨點遍歷一次，設定 $F = 10$，則我們會取出總共 $10N$ 個配對，

doi:10.6342/NTU202000440

其中 N 為群集圖中突出點個數。適當地限制 F，能夠限制雜湊值的大小，也同時降低資料庫的空間大小，但是 F 過小也容易造成從雜湊值中取出來的值無法凸顯出正確的歌曲，這我們將在 2.1.3 中說明。

2.1.3 比對及評分

完成特徵抽取的演算法與建立資料庫之後，下一步就是比對查詢片對與資料庫內的歌曲，藉由相同的演算法抽取特徵、比對，最後輸出評分分數最高的結果。該篇論文提出了一個複雜度逼近 $NlogN$ 的演算法進行比對，N 將在稍後給出定義。當查詢片段經過特徵抽取以後，會形成一張屬於查詢片段的雜湊表，接著我們將每一個屬於查詢片段的雜湊鍵與資料庫中的雜湊鍵進行比對，鍵值相同的情形下，我們會得到一個查詢片段的時間點 (t_k) 以及一個資料庫中該雜湊鍵對應的雜湊值中包含的所有時間點 $(t_k\prime)$ ，將 t_k 與各個 $t_k\prime$ 一一進行配對形成一個 $(t_k\prime, t_k)$ 的集合，將該集合映射到二維平面上，以雜湊值中包含的所有時間點 $(t_k\prime)$ 為橫軸，查詢片段的時間點 (t_k) 為縱軸，我們可以得到一個在特定雜湊值下的時間組合分佈圖 (scatter plot) ，分佈圖上點的個數即為 N 。若查詢片段對應到資料庫某歌曲，則這些對應到的特徵所形成的時間組合應該會呈現階梯狀的分佈，關係如下所示：

$$t_k\prime = t_k + offset \tag{2.1}$$

這樣的分佈展現在時間組合分佈圖上，會表現出一條斜率為 1 的斜直線。對於這條線上的每個點 $(t_k\prime, t_k)$ ，我們計算每點的時間差亦即縱軸－橫軸的差值 (δt_k) ：

$$\delta t_k = t_k\prime - t_k \tag{2.2}$$

而算出來的時間差應該是一個固定常數。

查詢片段是由使用者輸入的，因此它會包含可加性雜訊 (additive noise) ，這些雜訊具有高度不確定性，因此時間組合分佈圖上的點不一定會全部剛好組合為

doi:10.6342/NTU202000440

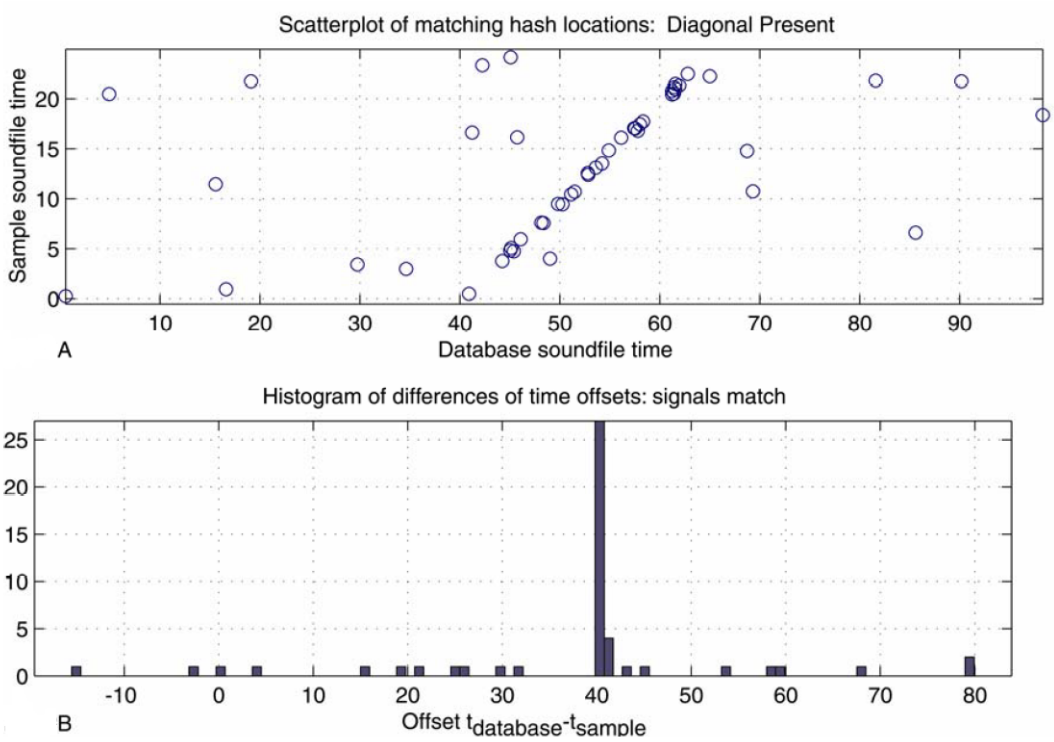

圖 2.3: 配對成功的時間組合分佈圖及時間差次數統計圖 [1]

一斜直線，部分點有可能零星散佈在直線之外如圖 2.3.A 所示，這時我們依然可以藉由統計時間差值的次數分佈來找出次數最高的值，如圖 2.3.B 所示，再由這個次數最高的時間差直往回追溯，得到所對應的資料庫時間點 $(t_k\prime)$ 以及其所在的雜湊值，而在 3.2.3 中我們提到過，雜湊值是一組集合，其中的元素是由歌曲編號與錨點時間所組合而成，由此我們可以找到每個資料庫時間點對應的歌曲編號，而該歌曲的評分就是對應時間差的統計次數。

如果配對失敗，那麼時間組合分佈圖便會呈現散狀，如圖 2.4.A ，時間差的次數統計也會趨近於平均分佈，如圖 2.4.B ，無法把正確的歌曲突顯出來，這也是我們上一小節中所提到，若是峰點數量太少則有可能會遇到這樣的困境。

doi:10.6342/NTU202000440

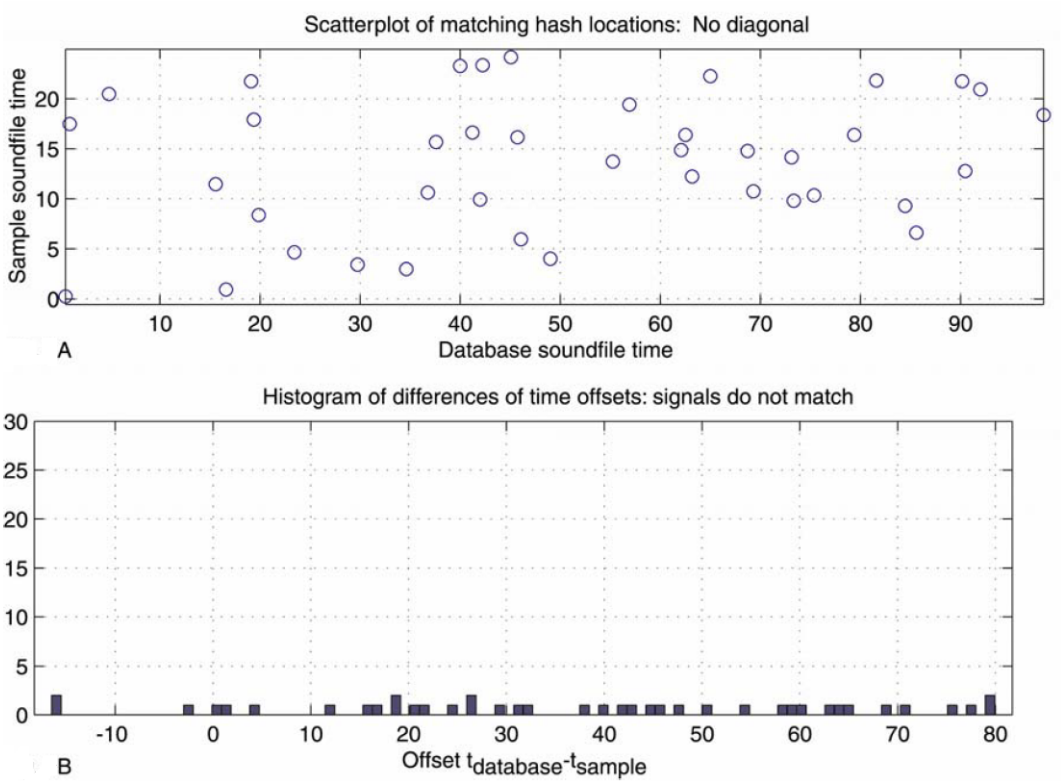

圖 2.4: 配對成功的時間組合分佈圖及時間差次數統計圖 [1]

2.2 Kim's Method

此方法是由 Hyoung-Gook Kim 於 2016 年發表 [2]，是一種基於 Wang's Method 的進一步改進，在抗干擾能力以及辨識率上比 Wang's Method 來得更加出色。以下將針對此篇論文改進的部分進行解說。

2.2.1 從 STFT (Short-time Fourier transform) 到 MCLT (Modulated complex lapped transform)

不同於 Wang's Method 使用 STFT 來對訊號做頻譜轉換，本篇使用 MCLT 來對訊號進行處理。首先利用 Hanning window function 將取樣過後的訊號切分成不同音框 (frame)，音框與音框彼此之間互相重疊，前一個音框的後 50% 的取樣值是後一個音框前 50% 的取樣值。接著利用式 2.3 進行 MCLT 轉換，其中 s 是原始

doi:10.6342/NTU202000440

訊號，S_M 是轉換過後的訊號，$U(k,l)$ 是快速傅立葉轉換 (fast Fourier transform)，$b(k,l)$ 是乘數因子，$h(n)$ 是大小為 N 的 Analysis window function，如下所示：

$$S_M(k,l) = |jV(k,l) + V(k+1,l)| \tag{2.3}$$

$$V(k,l) = b(k,l) \cdot U(k,l) \tag{2.4}$$

$$U(k,l) = \sqrt{\frac{1}{2N}} \sum_{n=0}^{2N-1} s(n+lO)h(n)exp(\frac{-j2\pi kn}{N}) \tag{2.5}$$

$$b(k,l) = W_8(2k+1,l) \cdot W_{4N}(k,l) \tag{2.6}$$

$$W_T(k,l) = exp(\frac{-j2\pi k}{T}) \tag{2.7}$$

在這裏 N 是取樣值的總數，k 是頻段的索引值 (index of frequency bin)，l 是音框的索引值 (index of frame)，O 是每一個音框中取樣值的總數。MCLT 的頻譜解析度 (frquency resolution) 是快速傅立葉轉換兩倍，因此藉由 MCLT 取出的峰點更加準確，不容易受到噪音、迴音或者解調 (demodulation) 失真所影響。

2.2.2 利用 Cosine Similarity 判別重複的峰點

如上一小節所述，對訊號進行 MCLT 轉換之前會先將訊號取樣並切分至彼此重疊的音框，這樣處理雖然可以降低訊號失真的程度，但是也會有大量重複的訊號取樣值被 MCLT 取出成為峰點，因此該篇論文使用 Cosine Similarity，$C(l_a, l_b)$，來判別兩個鄰近音框 $S_M(k,l_a)$ 和 $S_M(k,l_b)$ 中，其重疊部分的重複峰點，如式 2.8 所示。

doi:10.6342/NTU202000440

$$C(l_a, l_b) = \frac{\sum_{k=1}^{K} \quad S_M(k, l_a) \cdot S_M(k, l_b)}{\sqrt{\sum_{k=1}^{K} S_M^2(k, l_a)} \cdot \sqrt{\sum_{k=1}^{K} S_M^2(k, l_b)}} \tag{2.8}$$

式 2.8 計算結果為一個二維矩陣，裡面每個元素代表對應索引值的兩個音框之間的 cosine similarity ，接著將具有高度相關性 (cosine similarity 趨近於 1) 的所有音框進行濾波，由於我們要同時針對頻域與時域進行濾波而且目標是被視為雜訊的重複峰點，所以採用二維中值濾波器 (two-dimensional median filter) ，留下來的峰點用來當作辨識基底，被濾掉的峰點則是當作雜訊，如圖 2.5 所示。圖 2.5.a 是未經雜訊干擾的原始音樂訊號經過 MCLT 轉換後的頻域－時域群集圖；圖 2.5.b 是原始音樂訊號參雜了 SNR 6 dB 的雜訊後經 MCLT 轉換得到的頻域－時域群集圖；圖 2.5.c 是從圖 2.5.b 藉由 cosine similarity 以及二維中值濾波器處理過後的頻域－時域群集圖；圖 2.5.d 則是圖 2.5.b 經過二維中值濾波器後被濾掉的雜訊訊號所形成的頻域－時域群集圖。

2.2.3 利用雜湊表處理 Time-stretching 以及 Pitch-shifting

Time-stretching 以及 Pitch-shifting 是 Wang's Method 中未被討論及處理的部份，不同於 Wang's Method 中組成雜湊表的方法，這邊利用 start-to-end frequency ratio (SER) 來定義雜湊表，如下所示；

$$hash = (\varepsilon \cdot \frac{k_a}{k_p}, \theta \cdot \frac{k_a - k_p}{k_a}, (l_a - l_p) + (k_a - k_p)) \tag{2.9}$$

其中 k, l 分別代表頻域－時域群集圖上任一點的頻率以及時間，a 代表錨點 (anchor point) ，p 代表目標區域 (target zone) 中與錨點配對的突出點 (salient peak) ，$\frac{k_a}{k_p}$ 代表 start-to-end frequency ratio (SER) ，$\frac{k_a - k_p}{k_a}$ 則是 extent-to-start frequency ratio (FSR) ，這兩項能夠將一首經過 Time-stretching 或者 Pitch-shifting 的歌曲片段與原曲連結，降低 Time-stretching 以及 Pitch-shifting 帶來的識別錯誤，但同時也造成

doi:10.6342/NTU202000440

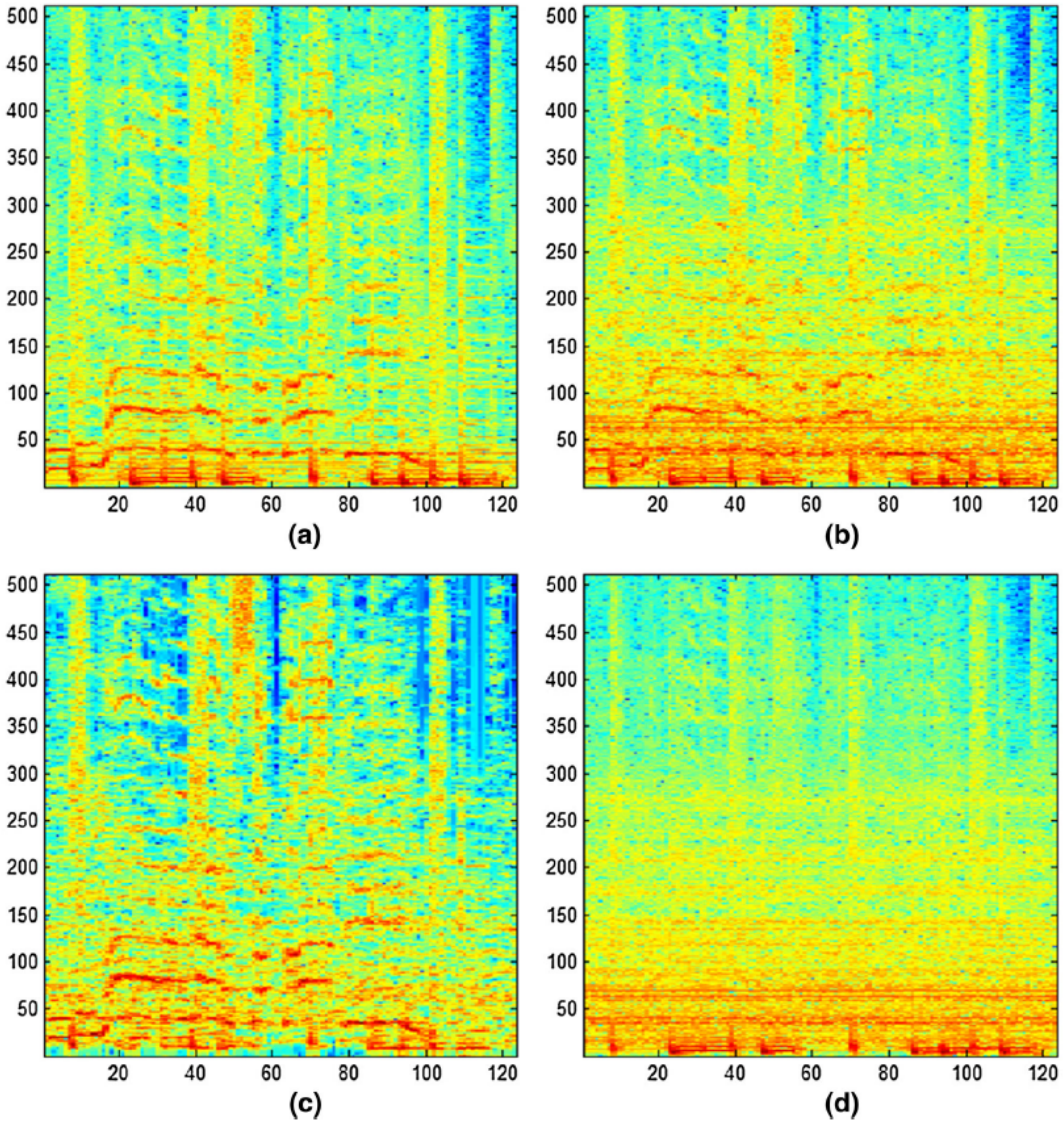

圖 2.5: 訊號處理過程中各個階段的頻域－時域群集圖 [2]

一個問題，SER 與 FSR 都是比值，所以組合而成的雜湊鍵分佈會非常集中，不僅降低辨識率也降低了辨識速度，為了解決這個問題，該論文使用不同的權重因子 (weighting factor) ，ε 以及 θ 來將雜湊鍵的分佈打散。

2.2.4 比對及評分

與 Wang's Method 類似，雜湊表建構完成以後，該篇論文將歌曲編號 (Song ID) 、地標 (Landmark) 兩點之間的時間差以及頻率差紀錄進資料庫，之後進行與

doi:10.6342/NTU202000440

Wang's Method 相同的比對過程，比對過程中，相同雜湊鍵的情況下，該篇論文會計算並儲存資料庫歌曲與查詢片段之間時間差的差值與頻率差的差值，並計算其總和 (T and F)，如下所示：

$$T = \sum (\Delta t_{DB} - \Delta t_{Query}) \tag{2.10}$$

$$F = \sum (\Delta f_{DB} - \Delta f_{Query}) \tag{2.11}$$

並分以下三情況討論：

- 該辨識片段並未經過 Time-stretching 或者 Pitch-shifting ，亦即用 Wang's Method 的比對方法也能得出可靠的辨識結果，將這種情況下所得到的 T 與 F 將當作基準。
- 該辨識片段經過 Time-stretching ，觀察這種情況下所得到的 T 與 F 會發現 T 比基準來得大，F 比基準來的小，這邊捨棄 T 值不予計算，只觀察計算 F 時出現次數最多的歌曲編號是多少，並找出對應的歌曲資訊回傳給使用者。
- 該辨識片段經過 Pitch-shifting ，觀察這種情況下所得到的 T 與 F 會發現 F 比基準來得大，T 比基準來的小，這邊捨棄 F 值不予計算，只觀察計算 T 時出現次數最多的歌曲編號是多少，並找出對應的歌曲資訊回傳給使用者。

doi:10.6342/NTU202000440

第三章　實作音訊指紋系統

3.1　系統架構及運作流程簡介

本論文所使用的實作方式是基於 Wang's Method [1] 的演算法，以及參考 Dan Ellis 提出的程式架構 [9]，更進一步改良衍伸而來。我們曾於第一章的研究背景中提到，現今知名的線上音樂辨識服務 Shazam [6]，其理論基礎即是 Wang's Method ，再進一步發展為商業應用。本章將闡述各個步驟的實作方式與相關細節，包含：如何抽取地標 (Landmark)、建構雜湊鍵 (Hash key) 與雜湊值 (Hash value) 並產生資料庫、查訊片段如何與資料庫進行比對並得出評分、最後輸出評分最高的歌曲編號與曲名。

圖 3.1 即為 AFP 系統的組成結構與音訊辨識的運作流程，其中分為兩大部分，分別是 $Offline : BuildingDatabase$ 負責線下運算建立資料庫，以及 $Online : MusicRecognition$ 負責線上與使用者交互，辨識查詢片段並回傳結果給使用者。在建構資料庫的部分，我們先將所有的歌曲原始音訊檔案分別抽取其地標 (Landmark)，再將地標轉換成雜湊鍵與雜湊值並建構雜湊表，最後將資料庫寫入檔案中以供之後讀取使用。而在音訊辨識的部分，使用者藉由手機、電腦或其他錄音設備錄製大約 10 秒的查詢片段，輸入進系統中，為了在有限的片段中盡可能獲取資訊，這邊會將抽取地標的流程重複進行四次取出大量地標，之後刪除重複的地標並將剩下的地標轉為雜湊表，最後再將此雜湊表與資料庫進行比對，根據時間差取得最終評分，依據評分對所有可能的歌曲進行排序，取前 10 首相似度相對高的歌曲作為結果回傳給使用者。以下各個小節將分別說明以上步驟的細節與做法。

doi:10.6342/NTU202000440

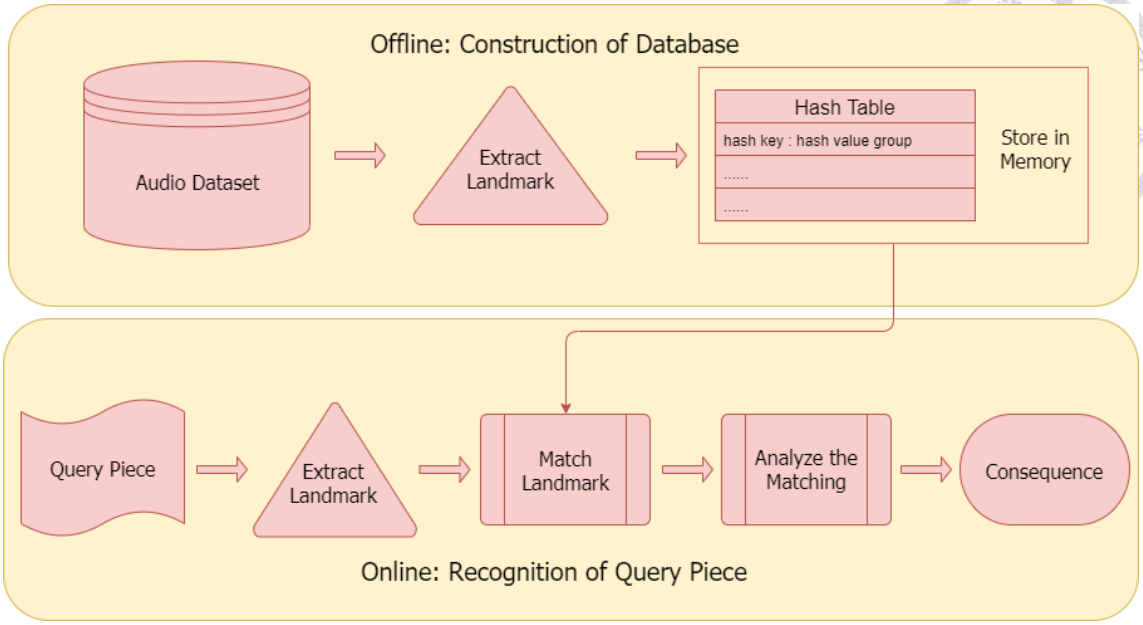

圖 3.1: 系統架構與辨識流程

3.2 抽取音訊特徵作為地標

地標是音訊指紋系統的核心，組成地標的內容將會大大影響系統辨識的準確率。抽取特徵及建立地標的流程如圖 3.2 所示，首先將一段音訊輸入，藉由短時距傅立葉轉換將其從時域-振幅軸轉換成時域-頻域-能量軸的頻譜圖，而後我們利用高斯衰減，先由前至後對時間軸進行篩選，選出區域極大值，再對這組區域極大值依時間軸由後至前進行一次高斯衰減，經過篩選後留下來的點即為突出點 (salient peak) 。對於每個突出點選取特定範圍當作目標區域 (target zone) ，在該範圍內的其他突出點依照能量大小依序排列，一開始的突出點，也就是錨點 (anchor point) ，依照此順序一一與之配對組成地標。

3.2.1 訊號前處理

在輸入音檔 (wav 格式) 的訊號值之後，若取樣頻率為 44,100 Hz，則對該訊號每 5 個點取 1 個點，將其降頻至 8,820 Hz。接下來每 1,024 個取樣值取作一個音框 (Frame) ，音框與音框之間重疊 50%，也就是前一個音框的第 513 到第 1024 個取

doi:10.6342/NTU202000440

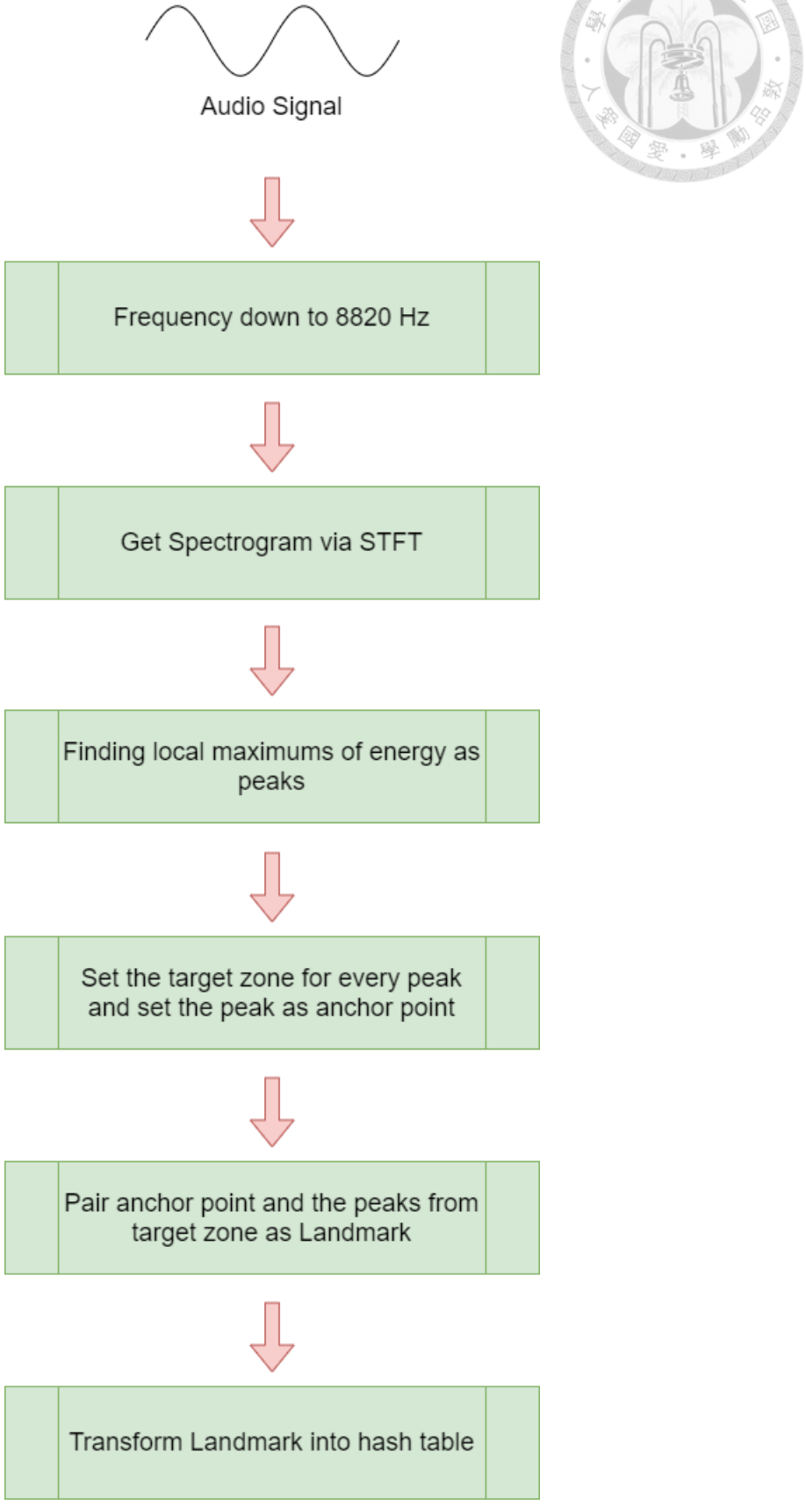

圖 3.2: 地標的建立流程

doi:10.6342/NTU202000440

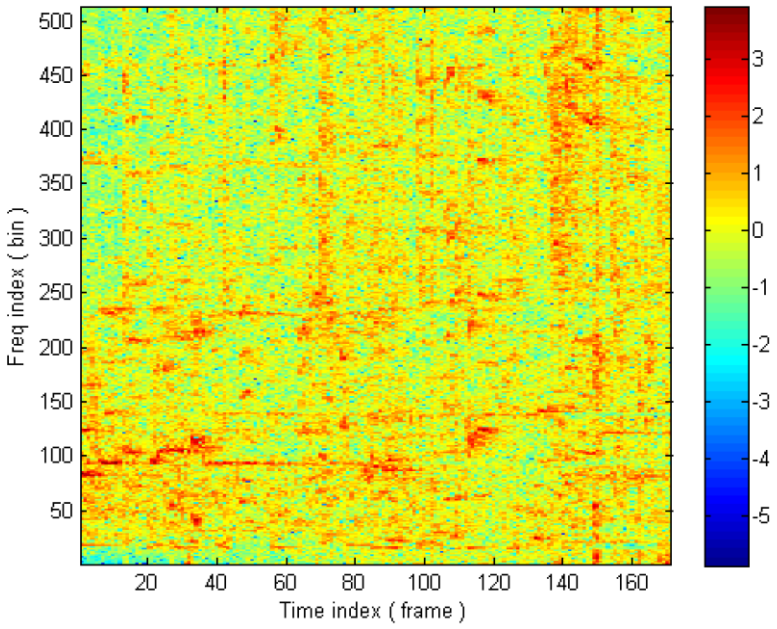

圖 3.3: 時域-頻域的對數頻譜圖

樣值會是後一個音框的第 1 到第 512 個取樣值，這樣的做法可以避免前後音框因為變化過於劇烈而導致失真，這裡也跟 Dan Ellis 所採用的做法 [9] 不同，Dan 採用的是將歌曲每 64 ms 分割成一個音框，每個音框之間重疊 50%，即 32 ms。接下來將每個音框進行傅立葉轉換，依時間排列，再將得到的數值取對數，為了降低頻噪音產生的影響，最後將每個頻段套入高通濾波器如式 3.1，形成時域-頻域的對數頻譜圖，如圖 3.3 所示

$$y(n) = x(n) - x(n-1) + 0.98y(n-1) \tag{3.1}$$

其中 x 代表頻譜圖中的頻帶，y 代表高通濾波器的輸出，n 代表頻帶的時間。

3.2.2 利用高斯衰減篩選突出點

承上節，我們會在對數頻譜圖上看到許多高能量的訊號點，我們將選出一些高能量的訊號點，它相較於周遭其他訊號點能量為最大，即為區域極大值，將這些點作為突出點。為了避免我們選出的突出點過於集中在圖中的某個區域，我們

doi:10.6342/NTU202000440

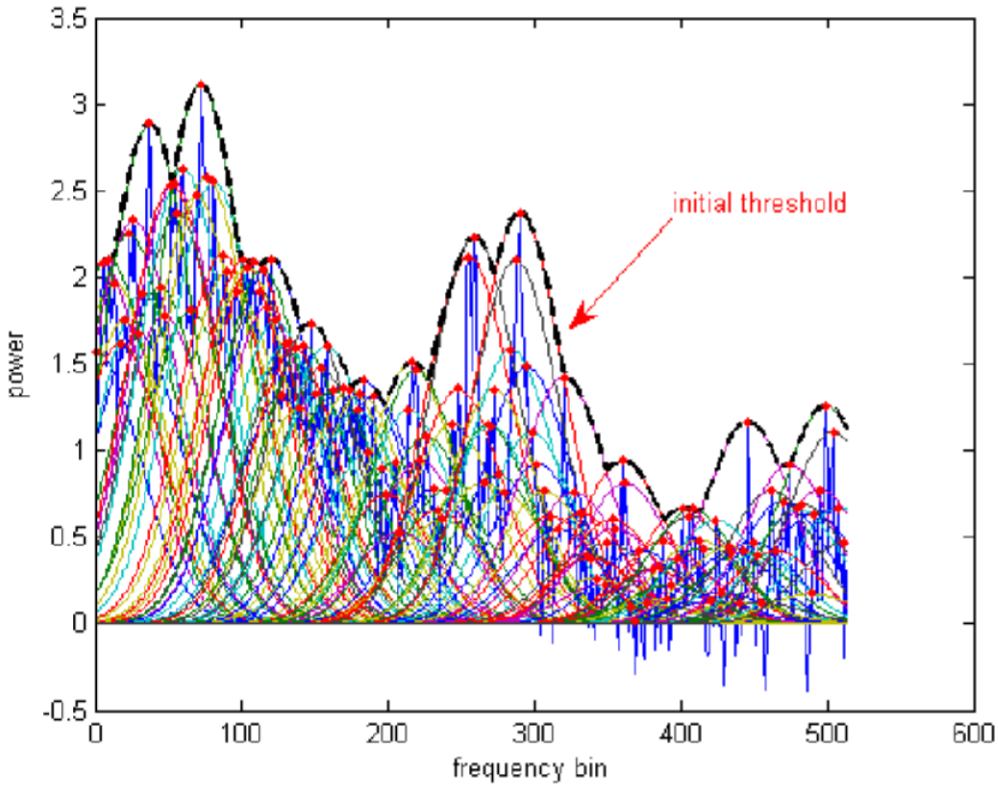

圖 3.4: 初始門檻值 [3]

必須動態調整能量的門檻值 (threshold value) ，因此我們用以下的流程定義在不同音框下的門檻值。首先我們取出前 10 個音框每個頻率上能量最大的取樣點並記錄其能量值，接著取出所有能量的區域極大值，即為峰點 (peak) ，以所有峰點為中心取高斯函數，將所有的高斯函數重疊定取其包絡，如圖 3.4中的黑色虛線，其中紅點為峰點，各色線條為取出的高斯函數。這條包絡便是門檻值的初始值。

接下來我們從第一個音框開始檢查，只要能量值大於門檻值我們便將其保留並記錄其能量值與門檻值的差，整個音框掃過以後，把所有已記錄的差值由大至小排序並只留下前 10 名，這 10 個點即為突出點，接著把門檻值乘上一個固定的衰減係數 $alpha$ ，這個衰減係數會介於 0 與 1 之間，衰減過後的門檻值如圖 3.5所示。回頭以 10 個突出點為中心取高斯函數，並與衰減過後的門檻值重疊，重新記錄所有重疊高斯函數的包絡，這便是下一個音框所使用的的門檻值，過程如圖 3.6及 3.7所示，取得新的門檻值後，如圖 3.7中的藍色虛線，就可以為下一個音

doi:10.6342/NTU202000440

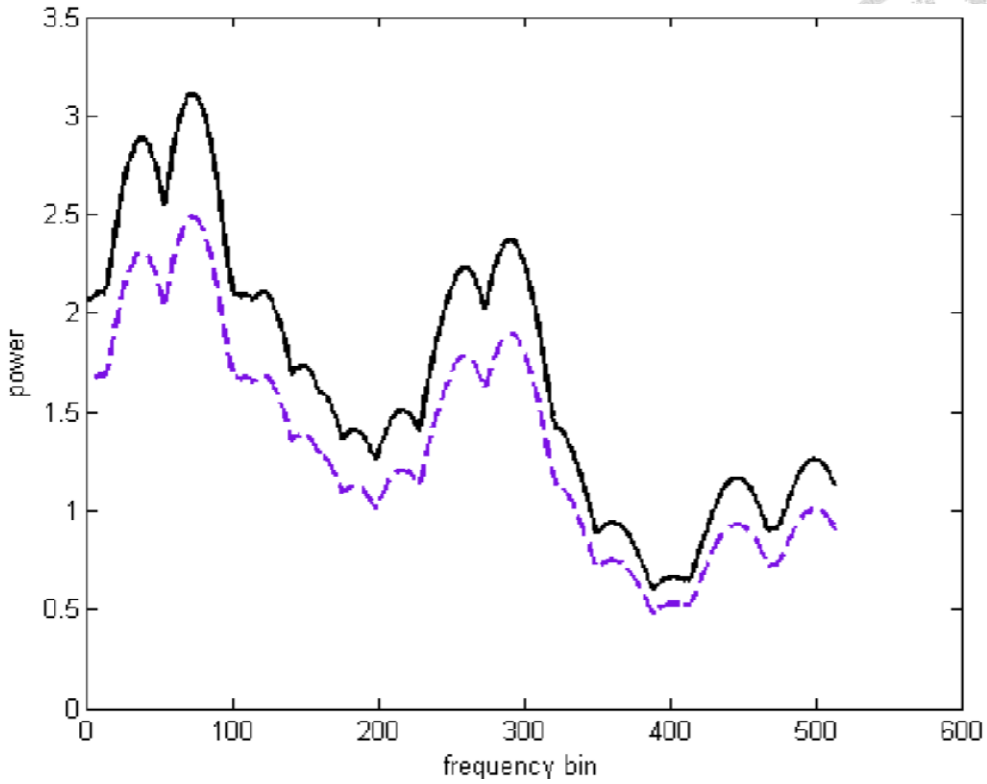

圖 3.5: 衰減過後的門檻值 [3]

框篩選突出點，循環執行此過程直到最後一個音框。這樣求取區域極大值以及紀錄高斯函數包絡的作法，可以避免突出點過度集中於某個頻段，而這些過於集中的突出點又會拉高該頻段的新門檻值，很可能造成下一個音框在對應的頻段無法取出突出點，如此一來在時域-頻域對數頻譜圖上，突出點會形成一個一個群落，無法均勻分散在圖上，這樣就無法記錄到該歌曲於所有時間點上各個頻率的特徵與內容，容易造成查詢片段無法正確找到對應的歌曲，降低辨識率。

在 Wang's Method 中，篩選突出點的步驟是正時序反時序要各做一遍，以保證篩選出來的突出點在兩種方向都是區域極大值，且分布足夠平均。這樣的作法篩去許多不合格的峰點，大大縮小資料庫的佔用空間，所抽取出來的歌曲特徵也不至於失之偏頗，識別率也有不錯的表現，然而在取樣頻率相同的情況下，比起單向篩選所花的時間，雙向篩選多付出了一倍的時間進行逆向篩選，而且是每一首歌曲或者查詢片段在做特徵抽取時，雙向篩選都要付出雙倍的時間，若是同時

doi:10.6342/NTU202000440

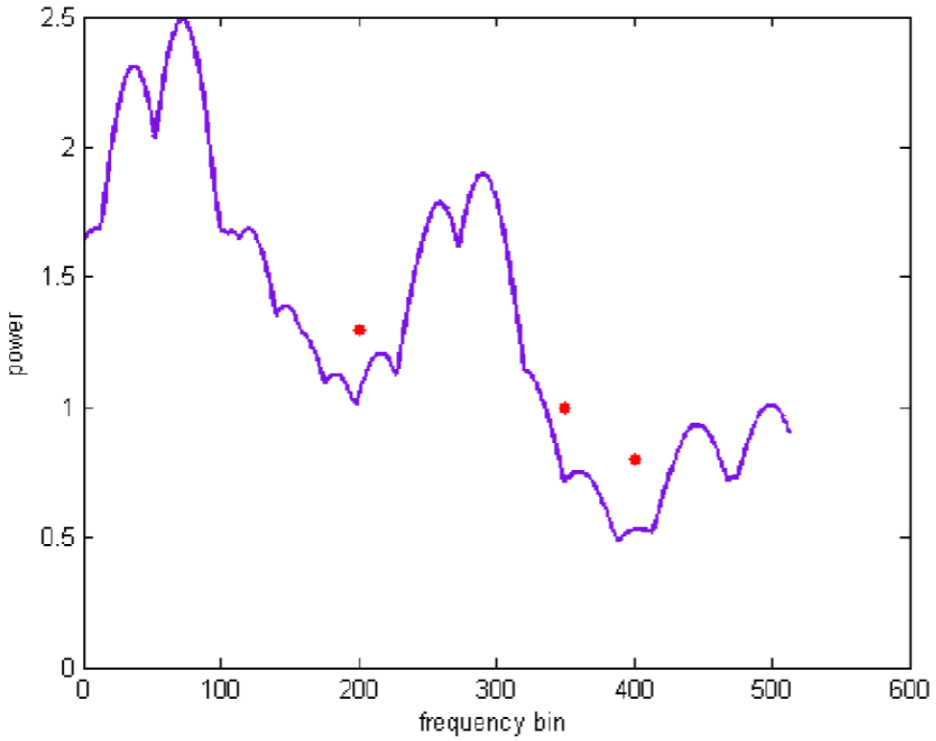

圖 3.6: 衰減門檻值與突出點 [3]

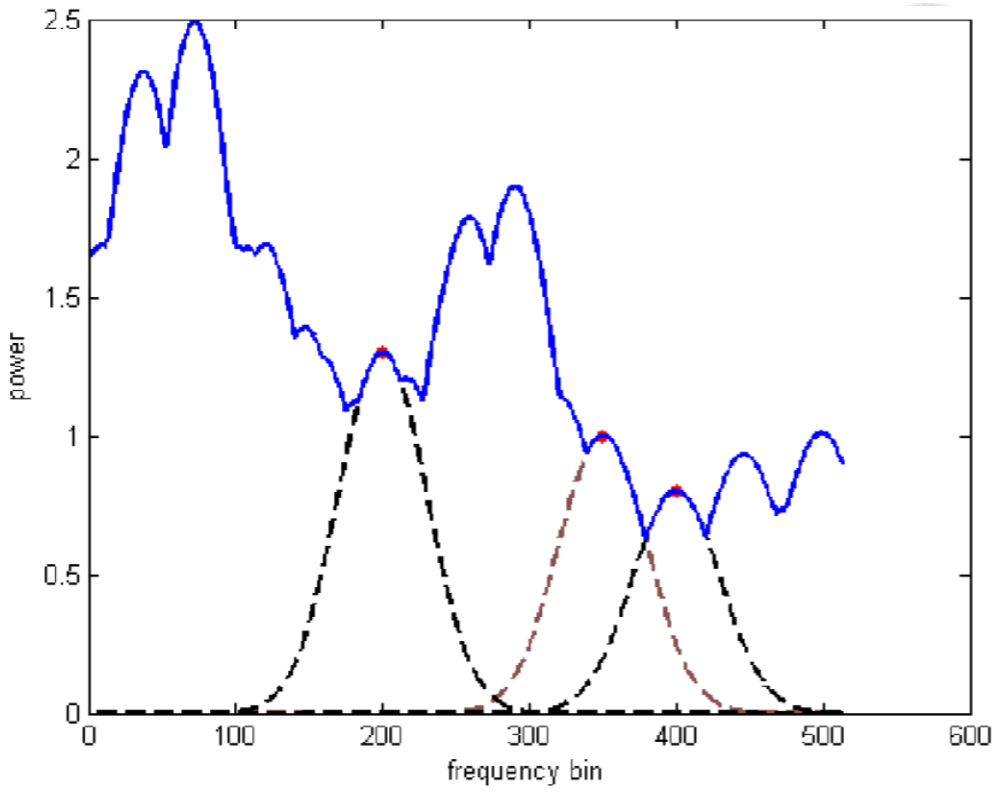

圖 3.7: 新的門檻值 [3]

doi:10.6342/NTU202000440

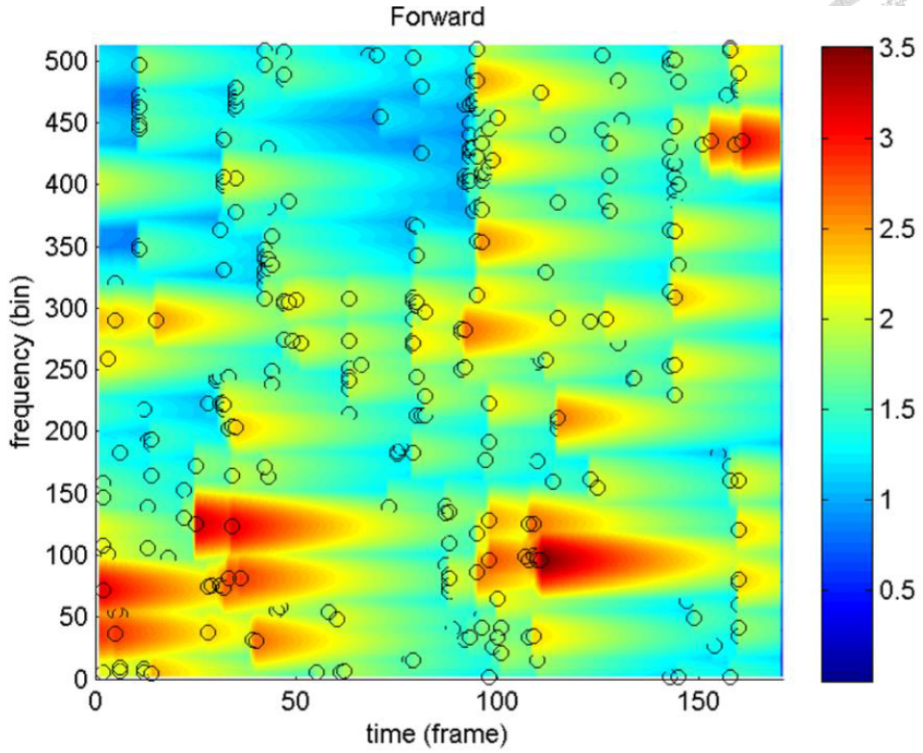

圖 3.8: 正時序篩選結果的投影 [3]

間過多查詢片段輸入，則會大大增加辨識到回傳結果所需要的時間。因此我們依循前人的研究 [3]，採用了與 Wang's Method 不同的方式，僅僅使用正時序的單向篩選。這樣的做法把辨識時間足足縮短了將近一半，但也產生了另外一個問題，篩選出來的突出點大大增加，在組合成地標的時候也會造成地標的數量爆炸性增長，如何解決這個問題我們將在 3.2.3 進行說明。

我們將每個音框的門檻值與突出點一起投影到時域-頻域對數頻譜圖，可以得到圖 3.8，圖中黑色的圓圈代表突出點，顏色代表能量高低，圖 3.8 是正時序篩選結果的投影，沿著正時序看過去，可以看到門檻值的能量並沒有隨著時間而不斷變大，這便是乘上衰減係數的效果，這樣可以避免時序後端取出的突出點越來越少甚至無法取出突出點，進而造成取出的特徵值無法充分呈現各個音框的訊號特徵。

doi:10.6342/NTU202000440

3.2.3 組合地標與建構雜湊表

在 3.2.2中，我們選出了一連串的突出點，下一步便是要將這些突出點所包含的資訊，組合成可以用來代表這首歌的特徵，也就是所謂的地標。循著正時序方向檢視每一個突出點，我們先選出第一個點當作錨點，以錨點為基準，依照給定的頻率與時間參數框選出目標區域，本篇論文採用的是以錨點的時間依正時序方向加 96 個音框以及錨點的頻率正負 128 個 frequency bin ，以此範圍作為目標區域，如圖 3.9 中的藍色區域所示，目標區域中每一個突出點都要跟錨點配對，並記錄每個配對的錨點時間 (t1) 、錨點頻率 (f1) 、配對中錨點與突出點的時間差 (Δt) 、配對中突出點的頻率 (f2) 。之後所有突出點都輪流當作錨點執行一遍同樣的流程，所得到的配對便是地標，如圖 3.9 所示。在 3.2.2 的後面我們提到只使用單向篩選造成的問題，突出點數量大大增加，經過配對後地標數量呈現爆炸性增長，為了限制地標的數量，我們根據前人的研究結果 [3] 採用了不一樣的參數。原來雙向篩選的做法中給訂了多組參數，按照前人的結論 [3] ，我們改動了其中兩組，第一組是每個音框的最大突出點數量 (maximum amount of peaks per frame) ，其值為 10 ，以及每個錨點最多能組成幾個地標 (maximum amount of pairs per peak) ，其值為 30 ，而我們採用的是每個錨點最多只能組成 8 個地標，不同於原本的 30 ；第二組是目標區域的時間跨度，原來採用的是 96 個音框，我們將目標區域左邊的邊界向右縮短 5 個音框，右邊的邊界向左縮短 60 個音框，如圖 3.10 所示，這樣可以大大減少地標的數量，而且前人的實驗結果 [3] 也表明在這樣的地標數量下，查詢片段的辨識率依舊表現良好。我們以集合 $S(t_i, f_i)$ 表示第 i 個錨點其目標區域中所包含的突出點 s ，其中 t 為時間，單位為音框，f 為頻率，單位為赫茲，如式 3.2 所示。

$$S(t_i, f_i) = \{s(t, f) \mid (t_i + 5) < t < (t_i + 36), (f_i - 128) < f < (f_i + 128)\} \quad (3.2)$$

doi:10.6342/NTU202000440

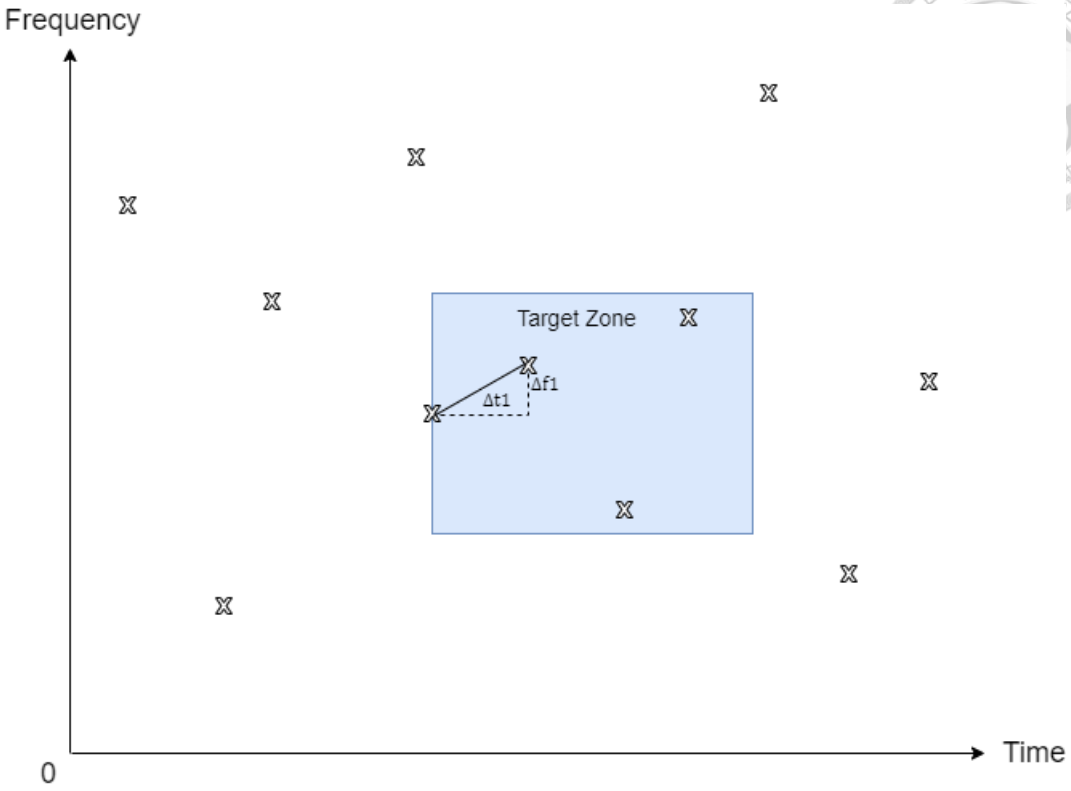

圖 3.9: 地標組成示意圖

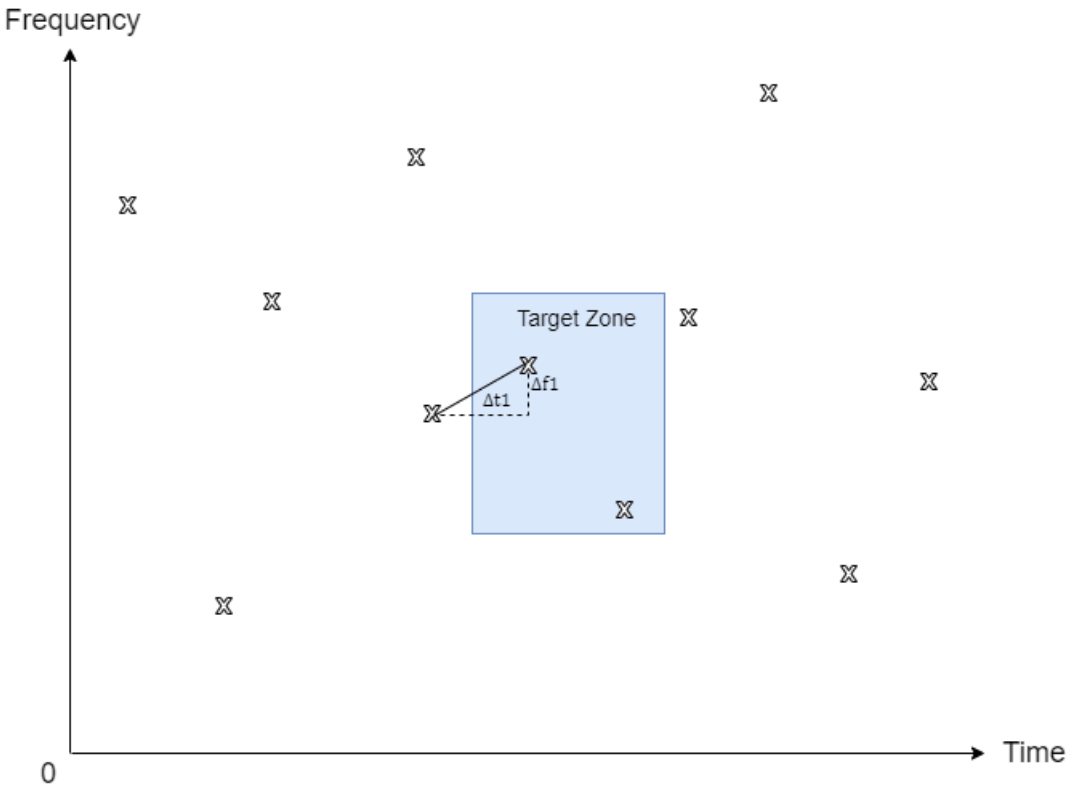

圖 3.10: 改動後之目標區域示意圖

doi:10.6342/NTU202000440

地標抽取完畢以後，為了加快檢索的速度以及增加比對的效率，我們使用雜湊表的形式儲存地標，上面我們紀錄了每個地標中配對兩點的頻率與時間資訊，t1 、f1 、Δt 、f2 四個數值。在雜湊表中，雜湊鍵的總長度為 24 位元，其中使用 9 個位元 (512 個 frequency bin) 紀錄錨點頻率 (f1) 。使用 8 個位元紀錄錨點與突出點的頻率差 ($\Delta f = f2 - f1$) ，其中 MSB (Most Significant Bit) 紀錄頻率差的正負號，0 為正，1 為負，後面 7 個位元紀錄頻率差的絕對值，範圍為 -127 +127，可以完整包含目標區域範圍內所有的突出點。最後使用 7 個位元紀錄配對中錨點與突出點的時間差 ($\Delta t = t2 - t1$) ，由於是正時序篩選，所以其值恆為正，目標區域給定的時間範圍為 96 個音框，因此僅需 7 個位元 (127 個音框) 即可完整包含目標區域範圍內所有的突出點。在雜湊值方面，歌曲編號 (Song ID) 代表資料庫中的每個地標其來源歌曲的編號，預設為 18 個位元，共可儲存 262144 首歌曲，位元長度愈長代表資料庫能夠儲存的歌曲數目愈多，這部分可依照資料庫中歌曲的數量進行動態調整，而 t1 為錨點時間，預設使用 14 個位元儲存，這部分位元長度代表資料庫中的歌曲所能容納的音框數目的最大值，1 個音框包含 1024 個取樣點，取樣頻率 8820 Hz ，單一音框時間長度約為 0.116 秒，又加上前後音框之間重疊 512 個取樣點，總共 2^{14} 個音框，換算為時間長度約莫 16 分鐘 ($0.116 \times 0.5 \times 2^{14} \div 60$) ，用來代表地標出處的代表值總共由以上 32 個位元所組成。圖 3.11 為雜湊鍵與代表值示意圖。為了處理那些長度超過 2^{14} 個音框的歌曲，我們定義了一個參數 Max Time 作為門檻，其值為 2^{14} ，當錨點時間 (t1) 超過 Max Time 時，我們會以取餘數的方式重新定義錨點時間，由於此時間亦為地標的起始時間，所以命名為 Start Time ，如式 3.3 所示。接著再以 Song ID 乘上 Max Time 後加上新的起始時間組成代表值，如式 3.4 所示。因為不同歌曲有可能取出相同的雜湊鍵，但是代表值並不同，直接用代表值當作雜湊值的話會形成一個一對多的鍵－值組合，這樣無法形成雜湊表，所以我們以雜湊鍵當作索引，所有對應到相同雜湊鍵

doi:10.6342/NTU202000440

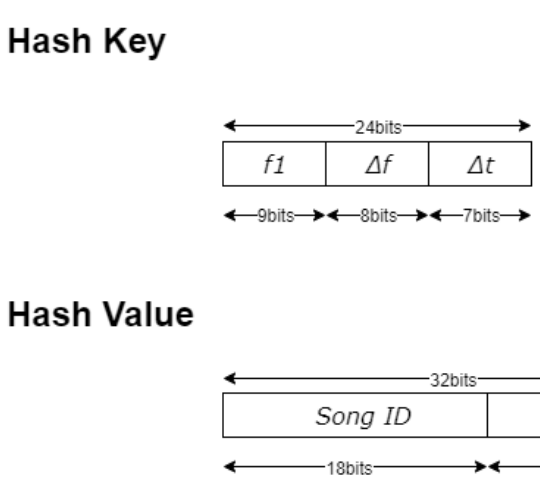

圖 3.11: 雜湊鍵與雜湊值

的代表值組合成一個集合，以這個集合當作雜湊值，組合為雜湊表。

$$Start\ Time\ =\ t1\ \ \mathrm{mod}\ (Max\ Time) \tag{3.3}$$

$$Hash\ value\ =\ (Song\ ID) \times (Max\ Time) + (Start\ Time) \tag{3.4}$$

3.2.4 以雜湊表建構資料庫

有了雜湊表之後，為了有效率的儲存雜湊表，我們將雜湊表分為兩個部分，並分別存入兩個二進制檔案。第一個部分是一個二維陣列，紀錄了所有的雜湊鍵以及每個雜湊鍵對應到的雜湊值中元素的個數；第二個部分則是所有的雜湊值，我們按照雜湊鍵的順序以及每個雜湊鍵對應到的元素個數，依序將雜湊值寫入檔案，讀取時也依照同樣的順序讀取，重新建構雜湊表。如圖 3.12 所示。如果有一組地標其雜湊鍵是 1 ，值是 1425 ，現在有另外一組地標雜湊鍵一樣是 1 ，值是

doi:10.6342/NTU202000440

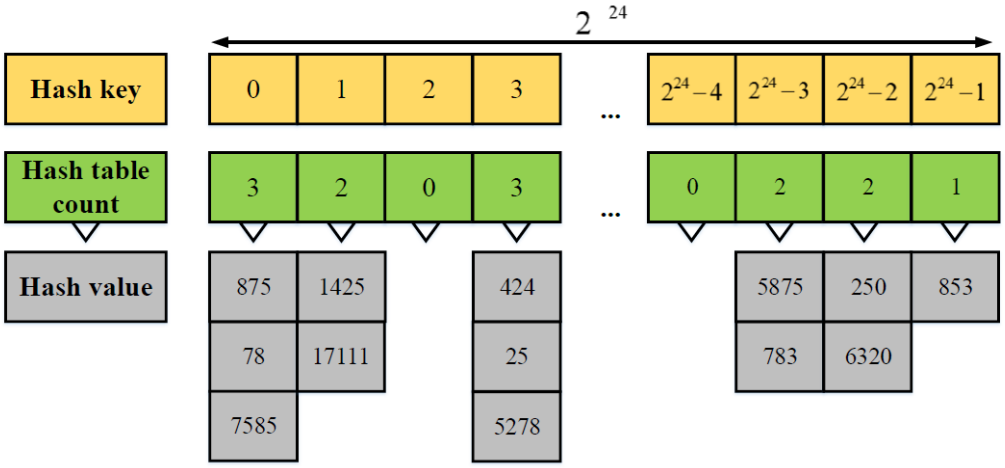

圖 3.12: 雜湊表

17111 ，則儲存的時候兩組值會組合成雜湊值，再跟雜湊鍵結合，構成完整的雜湊表，同時計數表中的元素個數加 1 。

3.3 辨識查詢片段

資料庫準備完成，我們接著要來建立系統架構中的線上部分，使用者輸入一段查詢片段後，我們首先抽取這段片段的特徵，接著將特徵轉換為地標並且建構出對應的鍵值與代表值，這邊我們並不會建立查詢片段雜湊表，查詢片段的鍵值是用來比對資料庫內的的雜湊鍵所用，當我們在資料庫的雜湊表內找到相同的鍵值時，便可以比對查詢片段的代表值與資料庫內的代表值，以此找出最有可能的偏移時間 (offset time) 並輸出最有可能的歌曲。以下我們將詳述其細節。

3.3.1 比對查詢片段與資料庫

系統接收到查詢片段後，首先用 3.2 中所述之方法，經過選取突出點、配對組合成地標、建構雜湊鍵與代表值，不同的地方在於，這裡的代表值內歌曲編號的所有位元皆為 0 ，因為我們還不知道查詢片段的對應歌曲，而後也不會建構雜

doi:10.6342/NTU202000440

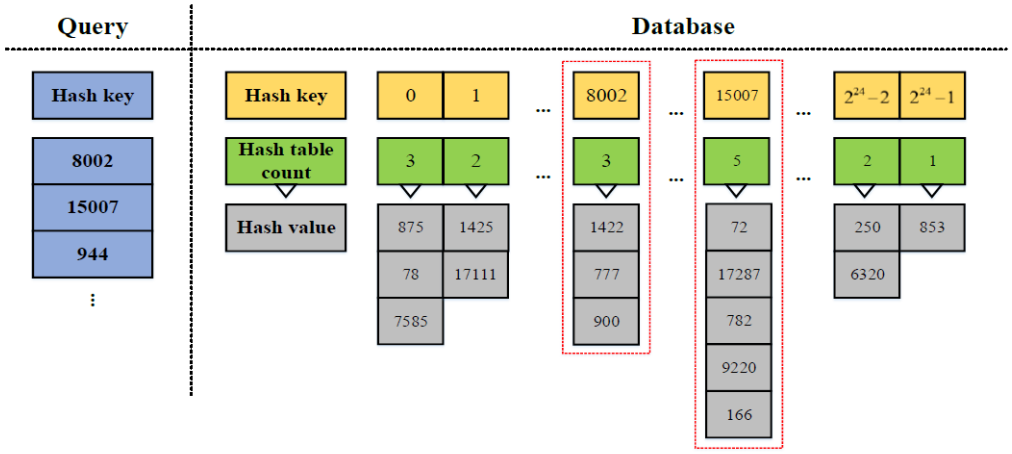

圖 3.13: 以辨識片段的雜湊鍵為索引從資料庫內取出雜湊值

湊表，代表值就等於該查詢片段地標的錨點時間 (t1) ，也就是該地標的起始時間。接著利用查詢片段的雜湊鍵為索引，直接提取資料庫內具有相同雜湊鍵的雜湊值，雜湊值內包含資料庫中所有對應相同雜湊鍵的代表值。以圖 3.13 為例，從查詢片段抽取出的雜湊鍵如圖中虛線左邊的區塊所示，虛線右邊則是資料庫的完整雜湊表，從查詢片段我們得到一系列雜湊鍵，其中以 8002、15007 為索引可以在資料庫中找到對應的雜湊值，如紅色虛線框住的部分，將所有雜湊值內的代表值一一取出，並記錄歌曲編號以及起始時間。

3.3.2 計算偏移時間 (offset time)

偏移時間的定義如式 3.5 ，我們以雜湊鍵為軸心，分別在查詢片段與資料庫中取出了各個代表值，接著把代表值中歌曲編號的部分獨立取出，先著眼於起始時間的部分，其中 $t1_{DB}$ 為資料庫中的起始時間，$t1_{query}$ 為查詢片段中的起始時間，兩者相減得到偏移時間。接著再將偏移時間與歌曲編號連結並記錄。

$$Offset\ Time = t1_{DB} - t1_{query} \tag{3.5}$$

如圖 3.14 所示，上方為資料庫中歌曲的完整片段，下方小長方形為查詢片

doi:10.6342/NTU202000440

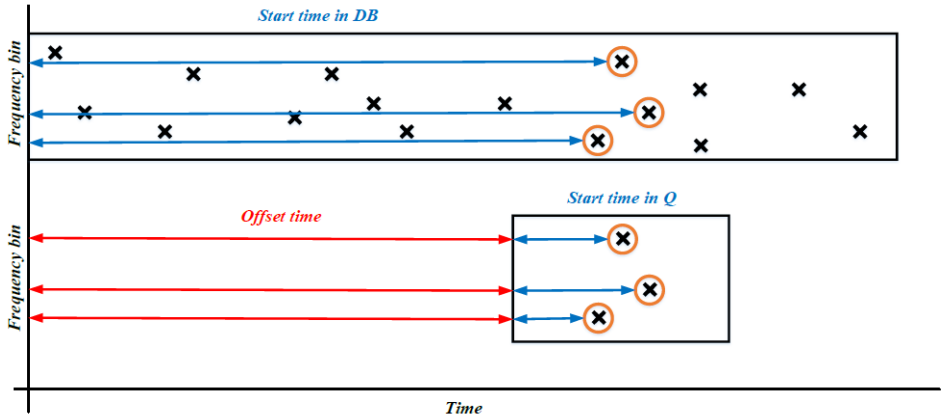

圖 3.14: 偏移時間示意圖

段，則雜湊鍵對應到的兩邊起始時間經由 3.5 ，可得到下方紅色雙箭頭所表示的時間差，此即為偏移時間 (offset time)。

3.3.3 統計偏移時間相同之歌曲與評分

在上一小節中，我們計算了偏移時間並紀錄了對應的歌曲編號，接下來要將所有歌曲編號與其偏移時間做分類與統計，如圖 3.15 。由於資料庫中每一首歌曲都有可能取出相同的雜湊鍵，所以每一個雜湊鍵底下所包含的歌曲編號不只一種，我們將偏移時間依歌曲編號整理在一起，先將圖 3.15 的結果按歌曲編號排序，如圖圖 3.16 ，接著對每一首歌曲統計每一種不同的偏移時間它的發生次數，統計時允許偏移時間可以有正負一個音框的誤差，累計該偏移時間正負一個音框內的次數，最後找出眾數作為該首歌曲的比對結果，亦是該歌曲的評分。以圖 3.15 的流程為例，最左邊的表按歌曲編號排序並記錄對應的偏移時間，也稱之為 hit list ，我們將編號 338 的歌曲所有的偏移時間取出，允許正負一個音框的誤差，將結果統計如中間的表，再選出次數最多的偏移時間眾數，以本表範例是 555 的 18 次為最多，最後將次數最多的歌曲編號取出，如圖中最右邊的表，這代表歌曲編號 338 的分數為 18 。

doi:10.6342/NTU202000440

Song ID	*Offset time*
154	74
487	3472
338	494
177	512
338	494
...	...

圖 3.15: 歌曲編號與對應之偏移時間

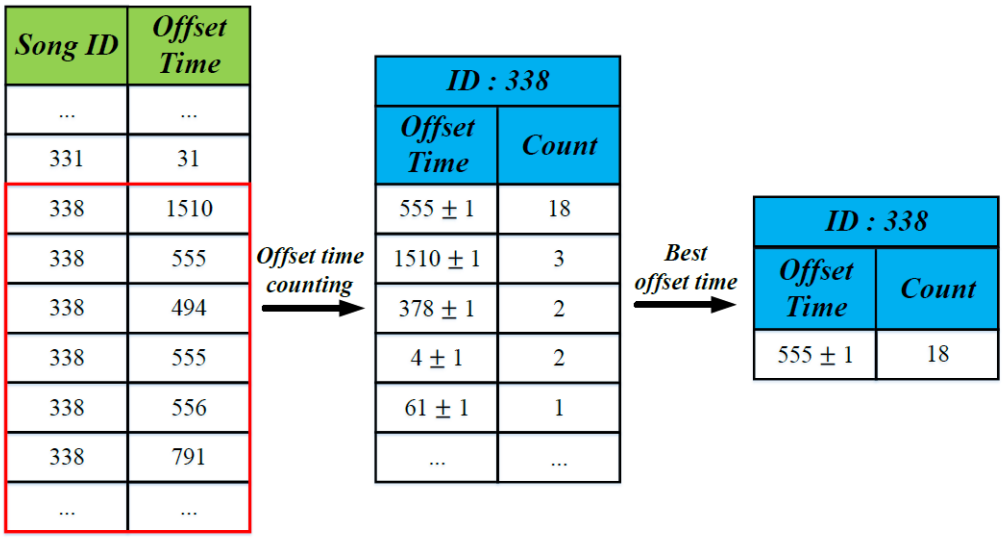

圖 3.16: 偏移時間的統計流程

doi:10.6342/NTU202000440

Rank	*Song ID*	*Matched landmark count*	*Offset time*
1	338	18	555 ± 1
2	154	10	10 ± 1
3	2240	10	158 ± 1
4	1047	8	2024 ± 1
5	680	5	1056 ± 1
...	...	...	...

圖 3.17: 分數 Top-10 排行

3.3.4 回傳分數前十高之歌曲資訊

根據上一小節的分數計算結果，我們取出分數前十高的歌曲並依分數高低排序，如圖 3.17 所示，根據這份排行榜的歌曲編號，取出歌曲相關資訊，將資訊與排名一併回傳給使用者作為辨識結果。在後續章節計算辨識率時，會只用第一名的歌曲當作最終辨識結果，亦即第一名的辨識結果與正確答案不符合即為辨識錯誤。

doi:10.6342/NTU202000440

第四章　改良方法與實驗結果分析

4.1　改良概念與測試環境簡介

4.1.1　改良目的與方法概念

AFP 音樂辨識系統在 Avery L. Wang 的設計之初便著重在辨識能力與抗干擾能力，其應對低頻以及中低能量雜訊時的確有很好的表現，但是當使用者所處的環境具有相當程度的噪音，像是夜市或者大賣場等地方，這類型的場所噪音主要來自於周圍的人聲，系統內的低通濾波器無法完全過濾，且能量之高甚至可以蓋過使用者想要錄製的音樂聲，因此系統在選取能量區域最大值作為突出點的時候容易選取到來自於噪音的訊號當作突出點，進而造成比對時雜湊鍵無法正確配對，評分結果也跟著失真導致辨識失敗。另外隨著 AFP 音樂辨識系統正式商業運轉，上線面對眾多同時湧入的使用者所輸入的查詢片段與辨識要求，辨識所花費的時間也成為一個重要的議題，除了透過平行運算的方式來加快雜湊表比對的過程以外，如何透過額外的資訊當作條件來減少比對的次數也是本篇論文的研究方向。有鑒於以上兩個問題，本篇論文將提出兩種改進方法分別針對兩個不同的問題。

首先針對辨識時間的部分，本篇論文藉由引進各個突出點的能量資訊，來當作除了時間與頻率以外的第三種參考，在雜湊鍵相同的情況下，組成地標的兩點，他們彼此間的能量大小關係會在查詢片段與原曲都呈現相同的趨勢，所以藉由兩點間能量關係來當作過濾條件，可以有效降低進入評分環節的 hit list 長度，減少評分所需的計算次數，達到降低辨識所需時間的效果。再來針對辨識率的部分，本篇論文提出一種對雜湊表進行雙向檢索 (bi-directional retrieval) 的方法，藉由正向檢索的方式得到歌曲編、偏移時間 (offset time) 與評分之後，將原來的雜湊

doi:10.6342/NTU202000440

表進行鍵－值反轉，以歌曲編號及偏移時間作為索引再進行一次檢索，我們稱之為逆向檢索，將原來的評分排名重新進行排序，最後將賦予新的分數不同的權重並與原來的分數相加，得到最後的評分結果，這樣的做法可以有效的將那些受雜訊影響而偏移的地標重新納入評分環節，讓這些一開始因為頻率偏移而無法正確與資料庫中的雜湊鍵配對的地標也能對辨識結果有正向的貢獻。最後利用排序學習演算法，以正向檢索以及逆向檢索各自所得出的評分當作特徵，針對排名進行學習，進一步提升辨識率。

4.1.2 測試環境

本論文所使用的實驗環境如下表所示，實驗分成單純使用 CPU 以及有 GPU 參與辨識的兩種模式，後續會介紹兩種模式的限制與異同。

OS	Ubuntu 18.04.3 LTS
Memory	128 GB
CPU	Intel(R) Xeon(R) CPU E5-2630 v4 2.20GHz
GPU	GeForce GTX 1080 Ti Memory: 11 GB
Programing language	C++ and Python
Database	10000 distinct songs from Mirex
Query pieces	5692 pieces in .wav format from Mirex Duration: 10 seconds Sample rate: 8820/s

表 4.1: 實驗使用之電腦配備、程式語言以及資料來源

doi:10.6342/NTU202000440

4.2 改進 GPU 平行運算所造成的辨識率下降

4.2.1 調整動機

本論文部分沿用了前人的研究成果 [3]，為了提高運算效率，使用 GPU (Graphics Processing Unit) 來進行平行運算，但同時也因為硬體限制以及演算法的瑕疵，使得辨識率比起單純使用 CPU 進行運算的演算法來的稍低，為了讓 CPU 版本與 GPU 版本可以在相同的立足點進行比較，我們使用下面的方法來調整原有的演算法。

4.2.2 調整方法

因為 GPU 中所含的運算核心 (core) 比起 CPU 多上許多，所以 GPU 進行平行運算最大的優勢便在於同一時間執行緒 (thread) 的數量比起 CPU 來的更多，首先我們觀察 GPU 版本，原來的演算法中，我們會先將查詢片段的每個雜湊鍵分到不同的區塊 (block) 中，每個區塊包含 512 條執行緒，用來處理該雜湊鍵從資料庫提取出來的雜湊值，最後得到一個完整的 hit list ，其中記錄了每次比對所得到的歌曲編號以及偏移時間，並以歌曲編號作為索引，如圖 4.1 所示。之後將 hit list 中每個歌曲編號分別配給一個區塊，每個區塊中 128 條執行緒，將每一首歌曲配對到的偏移時間進行排序，這裡使用奇偶排序演算法 (odd-even sort)，此種演算法利用平行運算實作相當有效率，利用 128 條執行緒先處理索引值為奇數的數字的比對與置換，再處理索引值為偶數的數字的比對與置換，演算法如 Algorithm 1 所示。根據演算法原理，128 條執行緒共可處理 256 的數字的排序，排序完成以後偏移時間便由小到大排列，相同的偏移時間也會落在相鄰的位置，如此便可以簡單地利用迴圈找出出現最多次的偏移時間是哪一個，再以這個偏移時間為基準逐個檢查所有的偏移時間，只要是偏移時間正負一的範圍內都予以計分，最後得到

doi:10.6342/NTU202000440

該首歌曲對本次查詢片段的配對分數。這時候出現第一個問題，原來的演算法中給定每個歌曲編號可以記錄的偏移時間數量最多為 1024 筆，但是使用 128 條執行緒的奇偶排序卻只能排好前 256 筆紀錄，後面 768 筆紀錄完全未經排序，如此後面進入評分迴圈時，所得出來的分數便會有誤，如圖 4.2 所示。第二個問題則是出在對 hit list 最大紀錄數量的限制上，這邊給定最大紀錄數量為 1024 筆，但是在進行雜湊表比對的時候，單一歌曲編號被比對到的紀錄數量很容易就超過這個數量限制，這也造成第 1025 筆以後的紀錄會被捨棄，如圖 4.3 所示，少了這些紀錄帶來的評分，最後給出的分數比起逐筆比對自然有所誤差，進而影響到辨識結果的正確率。為了解決這些誤差，我們將奇偶排序時單一區塊的執行緒數量拉到 1024 條，此數量亦為目前 GPU 平行運算所支援的上限，而根據奇偶排性的特性，我們也可以將單一歌曲編號的最大紀錄數量拉到 2048 筆。經過這樣的調整後，僅剩 1.97% 的查詢片段會具有超過 2048 筆的紀錄數量，但由於前面已經紀錄 2048 筆，剩下沒紀錄到的誤差對辨識率影響相對輕微。接著我們也對評分的判定條件進行微調，原本 Wang's Method 中的判定條件是偏移時間正負一的範圍內都算做同一種偏移時間並予以計分，我們將條件改為要偏移時間完全相同才予以計分，如圖 4.4 所示，並且在 CPU 跟 GPU 兩個版本中實作這個改動。調整的方法總結如下。CPU 版本：

- 將計分條件改為要偏移時間完全相同才予以計分。

GPU 版本：

- 將奇偶排序時單一區塊的執行緒數量提升至 1024 條。
- 將單一歌曲編號的最大紀錄數量提升至 2048 筆。
- 將計分條件改為要偏移時間完全相同才予以計分。

doi:10.6342/NTU202000440

圖 4.1: hit list

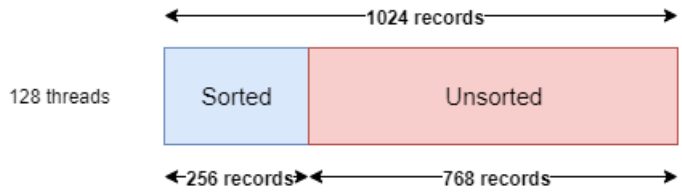

圖 4.2: 排序演算法造成的誤差

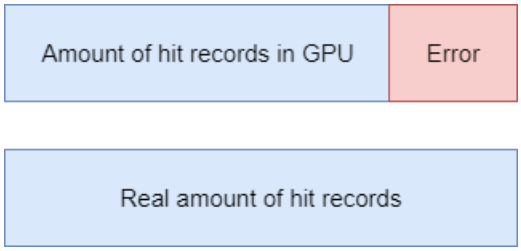

圖 4.3: 限制單一歌曲最大紀錄數量所帶來的誤差

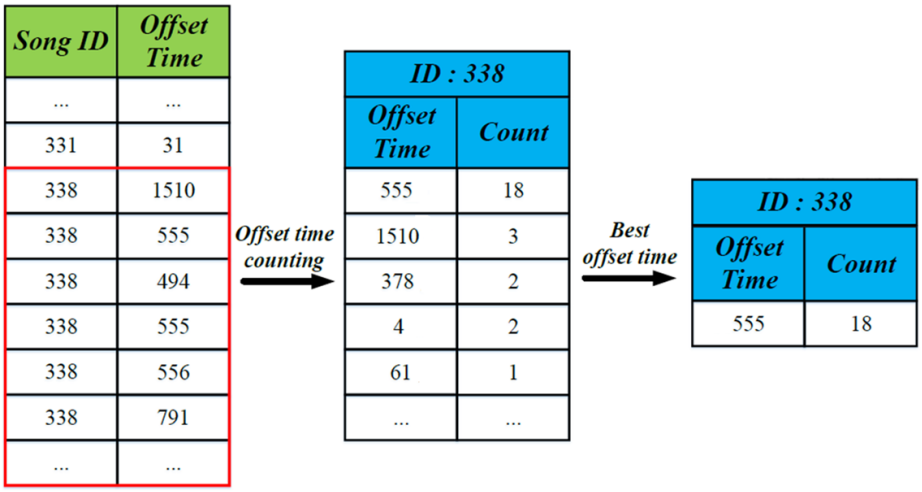

圖 4.4: 改動計分條件

doi:10.6342/NTU202000440

Data: hit list of one song, thread amount per block

Result: sorted hit list in the order of offset time

tid = thread ID;

while *(tid) < (maximum hit count per song)* **do**

    each thread loads one record from hit list into array Q;

    tid += thread amount per block;

end

for *i = 1 to hit count of one song* **do**

    t = (thread ID) * 2;

    if *(t+1 < hit count of one song) & (Q[t] > Q[t+1])* **then**

        swap Q[t] and Q[t+1];

    end

    t = t + 1;

    if *(t+1 < hit count of one song) & (Q[t] > Q[t+1])* **then**

        swap Q[t] and Q[t+1];

    end

end

Algorithm 1: 利用奇偶排序法將 hit list 依照偏移時間由小到大排列

doi:10.6342/NTU202000440

4.2.3 調整結果

表 4.2 以及 4.3 分別是調整前與調整後的系統表現，從資料庫的大小、平均辨識時間以及辨識率三個方面來做比較。比較兩張表的數據，可以看到 CPU 版本僅僅在計分條件經過改動以後，辨識率成長了 1.4% ，而 GPU 版本經過同樣的改動並且修正了排序演算法中的誤差以後，辨識率不但有明顯進步，而且兩個版本的辨識率誤差僅有 0.1% 。辨識時間上，因為我們將計分條件變得嚴格，所以需要進行計分的迴圈數量變少了，CPU 版本的平均辨識時間有少許進步，而 GPU 版本則是因為最大紀錄數量放大，也使得計分迴圈數量放大，所以辨識時間增加了 0.03 秒，但同時辨識率增加了 3.1% 。

	CPU Version	GPU Version
size of database (GB)	3.74	
average query time per song (second)	1.14	0.12
accuracy (%)	87.2%	85.6%

表 4.2: 調整前的系統表現

	CPU Version	GPU Version
size of database (GB)	3.74	
average query time per song (second)	1.12	0.15
accuracy (%)	88.6%	88.7%

表 4.3: 調整後的系統表現

doi:10.6342/NTU202000440

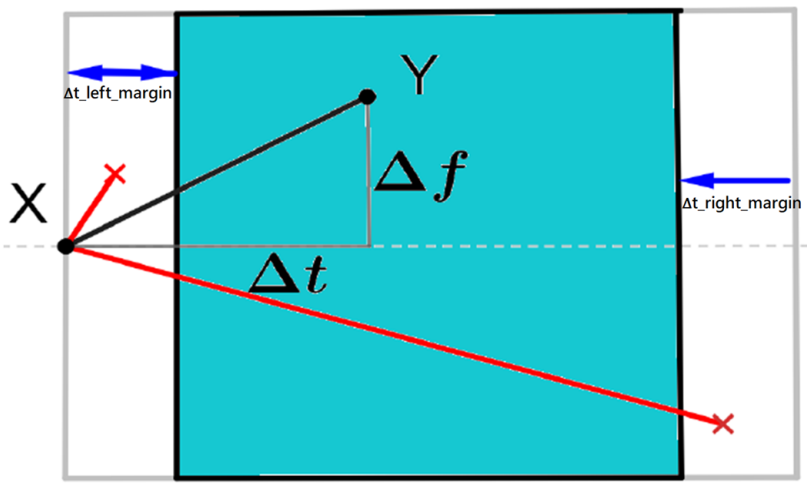

圖 4.5: 目標區域示意圖

4.3 建構複數雜湊表進行比對

4.3.1 改進動機

在第三章我們介紹過如何抽取一首歌曲的特徵並組合成地標，峰點經過動態的能量門檻值的篩選後，得到突出點，之後將每一個突出點輪流當作錨點並與目標區域內的其他突出點進行配對組合成地標，我們所使用的目標區域範圍與 Wang's Method 有些許不同，如圖 4.5 所示。我們取 $\Delta t\_left\_margin = 5$ 以及 $\Delta t\_right\_margin = 60$ ，所以每個錨點的目標區域其左右邊界為該點時間 t 後方的 t + 5 到 t + 36 的區間內，上下邊界為該點頻率上下各 128 個頻帶共計 255 個頻帶，我們利用兩點之間的時間差與頻率差來當作該地標的特徵，並利用錨點時間以及頻率來當作座標以標示該地標在歌曲中的起始位置，如果我們引進更多的有效資訊來建構多張雜湊表，對地標進行多層過濾，也許可以在辨識率不變的情況下降低辨識所需的時間。

doi:10.6342/NTU202000440

4.3.2 引進能量資訊建構第二張雜湊表

音訊指紋系統的研究已經行之有年，除了 Wang's Method 使用頻率以及時間來當作音訊特徵，也有許多使用能量來當作音訊特徵的音訊指紋系統，像是 Jijun Deng 以及 Xueqian Pan 等人於 2011 年所提出的設計 [10] [11]，因此我們這邊決定使用地標兩點間的能量差來當作第二張雜湊表的內容，我們的假設是:「任一地標的兩個突出點之間，其兩點間的能量關係會在原歌曲與查詢片段上呈現相同的分布趨勢」。根據以上的假設我們做出下面針對第二張雜湊表的設計原則：

- 新雜湊表是用來當作地標的過濾器，並且與原來的雜湊表來自共同的地標，所以兩張雜湊表共用相同的雜湊鍵。
- 由於能量容易受到環境雜訊的影響，因此在相同雜湊鍵下，只要地標內兩突出點的能量差具有相同的大小關係，並且能量差值在一固定區間之內，我們就將這個地標納入 hit list 中進行評分。

設計細節如圖 4.6 所示，雜湊表二在雜湊值的組成上，其所包含的代表值中使用了能量差取代錨點時間，總共使用 14 位元儲存，由於代表值的儲存進資料庫中的格式為 unsigned int ，因此我們將能量差的精度取到小數第二位，再轉換為 unsigned int ，第一個位元用來代表能量差的正負號，後面 13 個位元用來儲存能量差的絕對值。

雜湊表建構完畢以後，我們便要利用能量差來進行過濾，減少被納入 hit list 中的錯誤配對，以期能提升辨識率並同時降低辨識時間。過濾條件如式 4.1 所示。

$$[h](1-\rho)(e_1-e_2)_{DB} < (e_1-e_2)_{Query} < (1+\rho)(e_1-e_2)_{DB},\ 0 < \rho \leq 1 \quad (4.1)$$

$(e_1-e_2)_{Query}$ 代表查詢片段中的能量差，$(e_1-e_2)_{DB}$ 代表在相同雜湊鍵下，資料庫中對應的能量差值，我們以資料庫中的能量差為基準，查詢片段所得到的能量

doi:10.6342/NTU202000440

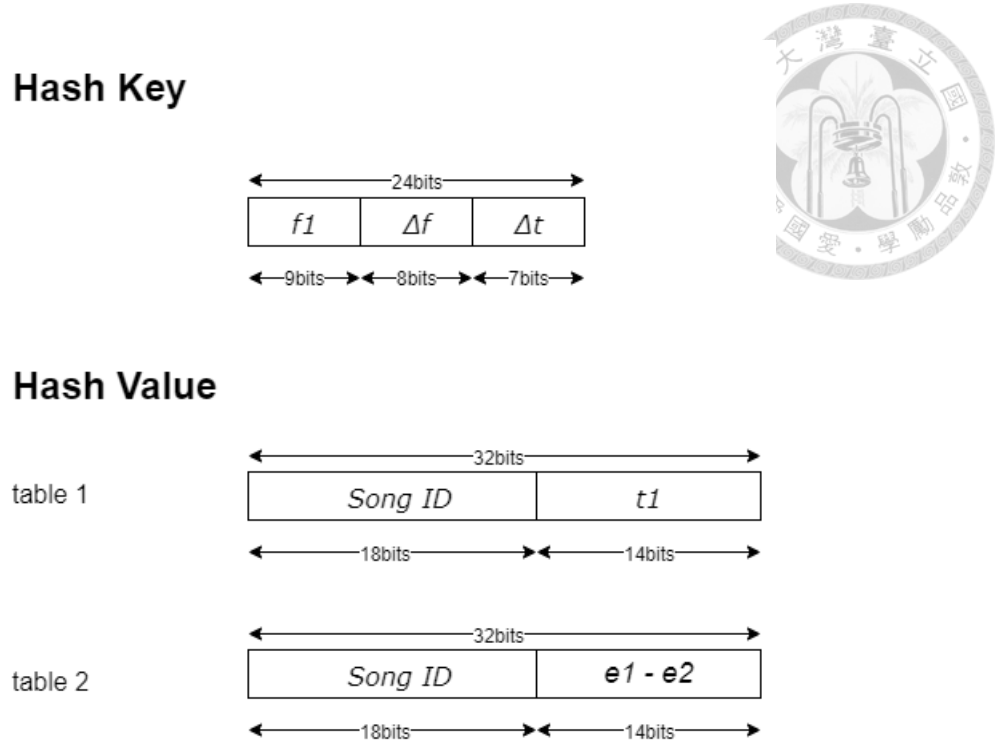

圖 4.6: 原雜湊表與新雜湊表示意圖

差若是落在 $[(1-\rho)(e_1-e_2)_{DB},\ (1+\rho)(e_1-e_2)_{DB}]$ 之間，便將資料庫中這筆能量差與其歌曲編號納入 hit list 。

4.3.3 改進結果與分析

本論文分別採用 $\rho = 0.1, 0.25, 0.5, 0.75, 0.9$ 進行測試，得到的結果如下圖 4.7 以及 4.8 所示，從圖中我們觀察到，使用能量來當作過濾地標的條件，會使的辨識率下降，ρ 越小，區間越小，則辨識率下降越多，但同時辨識時間也顯著下降，這代表使用能量進行過濾也會將帶有正確資訊的地標過濾出 hit list 。

我們在辨識率與辨識時間兩者之間取一個相對好的結果，如表 4.4 所示，在 $\rho = 0.5$ 的條件下，正確率從 88.6% 下降到 85.1% ，下降 3.5 個百分點，辨識時間從 1.12 秒縮短為 0.705 秒，縮短 37.04% 。即使我們捨棄使用能量差的區間作為條件，僅僅以能量差的正負號，也就是兩個突出點之間的量大小關係來當作過濾

doi:10.6342/NTU202000440

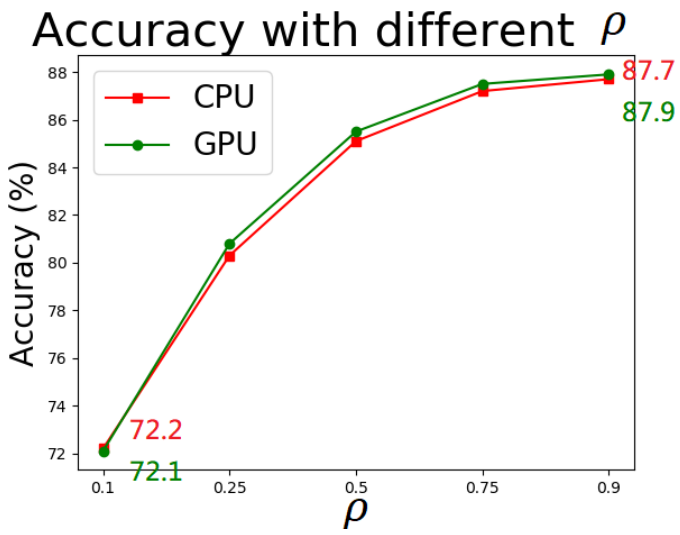

圖 4.7: 以能量作為過濾條件所得之辨識率

條件，辨識率也只有 87.7% ，依舊是比 3.2 節中得到的表現來的差。

	CPU Version	GPU Version
size of database (GB)	7.4	
average query time per song (second)	0.7051	0.0803
accuracy (%)	85.1%	85.4%
ρ	0.5	

表 4.4: $\rho = 0.5$ 的系統表現

接著我們針對那些造成辨識率下降的歌曲，亦即本來配對正確，經過過濾後反而錯誤的歌曲，來進行分析，首先觀察查詢片段與資料庫中原曲兩者之間的能量差關係，如圖 4.9 所示，橫軸為查詢片段與原曲兩者共有的雜湊鍵，縱軸為能量差值，可以明顯看出該查詢片段的能量分布嚴重受雜訊干擾而失真，進而無法與原曲正確地進行配對，而實際去聽該查詢片段，環境噪音嚴重影響樂曲，甚至幾乎聽不出樂曲的音調。由此可知此方法易受環境噪音影響，但從實驗結果來看，平均辨識時間縮短了 37.04% ，GPU 版本甚至縮短了 46.46% ，也就是說在

doi:10.6342/NTU202000440

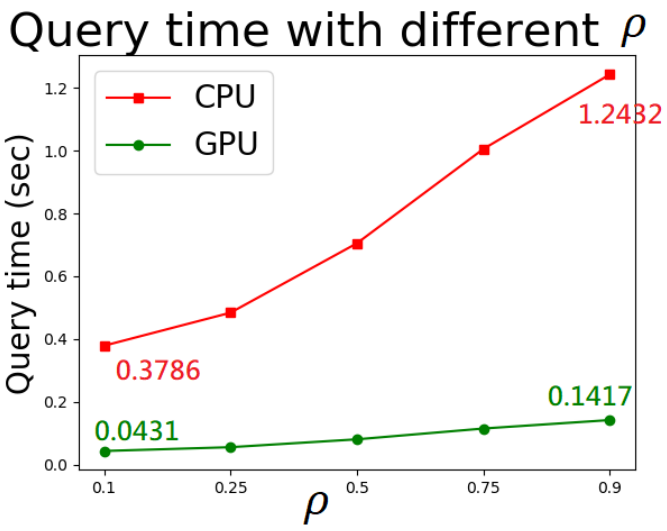

圖 4.8: 以能量作為過濾條件所得之辨識時間

$\rho = 0.5$ 的情況下，我們平均過濾掉了 37% 以上的地標，但同時辨識率僅下降 3.5%，這意味著在環境噪音對查詢片段的干擾不大的時候，查詢片段與原曲的確有相同的能量分布趨勢，而隨著噪音越加明顯，兩者能量分布趨勢的一致性也隨之被破壞，如圖 4.9 中所看到的極端例子，兩者的能量分布幾乎毫無關聯。

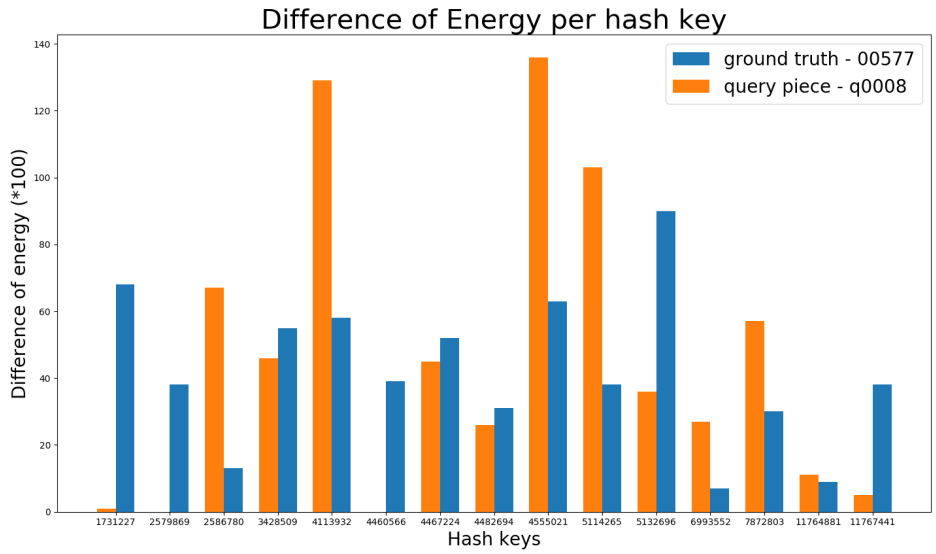

圖 4.9: 能量分布趨勢圖

doi:10.6342/NTU202000440

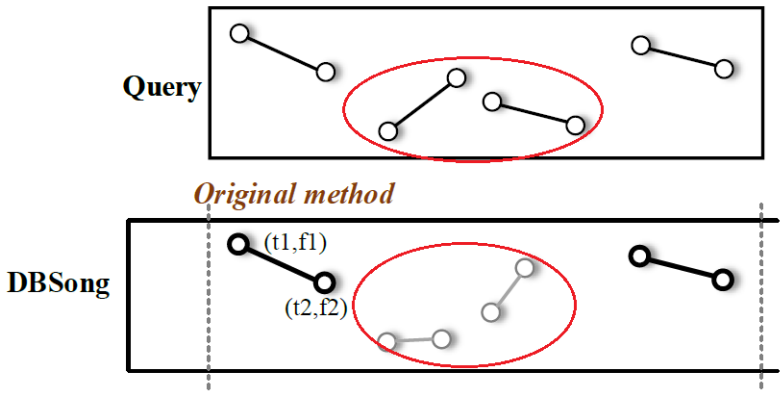

圖 4.10: 受噪音影響而造成地標不匹配

4.4 利用雙向檢索比對雜湊表

4.4.1 調整動機

在 Wang's Method 中，辨識方法是以雜湊鍵為第一層審核機制，也就是說地標中的 t1 、f1 、Δt 、Δf 四者都必須相同，接著再以偏移時間眾數的個數作為評分依據，這樣該歌曲才會獲得配對分數，這樣的條件雖然保證了辨識結果可以不受大部分雜訊的干擾，但也顯得較為嚴苛，如果有一個地標，它在原曲與查詢片段有相同的錨點頻率 (f1) ，但是查詢片段受到高能量噪音的影響導致頻率差 (Δf) 受到影響而有所偏差，也進而導致時間差 (Δt) 與原曲不同，那這樣一個地標就不會被納入 hit list 中進入評分環節，但事實上它所攜帶的資訊是對配對結果有正向影響力的，如圖 4.10 所示。為了改善這類型的失真，本篇論文將逆向對雜湊表進行二次比對，希望將具有相同錨點頻率、歌曲編號以及偏移時間的地標重新檢索一次，將頻率差落在一固定區間內的地標重新統計並評分，讓具有正向影響力的地標可以在這裡被納入考量。

doi:10.6342/NTU202000440

4.4.2 交換雜湊值與雜湊鍵進行逆向檢索

從上一小節的思考方向中，我們得到一個明確的目標，我們希望可以將受噪音影響而被排除在外而且對辨識結果具有正向影響力的地標重新納入評分，因此我們做出一個假設：「這些因為噪音而失真的查詢片段，其原曲雖然沒有被評為最高分數，若將歌曲依據分數排名，原曲的分數也應該會落在前列」。根據以上的假設我們得到以下的設計原則：

- 以歌曲編號、偏移時間以及錨點頻率作為索引，重新檢索一遍各個地標的頻率差，若落在一固定區間則將該地標納入評分。
- 由於資料庫中歌曲眾多，不可能將每一首歌曲都重新評分，僅取前 10-50 名進行重新評分。
- 最後賦予新的分數一加權因子，與原分數結合為最終評分。

首先來到第一步，反轉雜湊表，為了滿足第一條設計原則，我們必須利用原來的評分方法得到初始分數、偏移時間與歌曲編號。接著將原雜湊值中各個代表值轉換為雜湊鍵，而該代表值對應的原雜湊鍵轉變為雜湊值，如圖 4.11 所示。接著我們利用原評分方法給出的歌曲編號與偏移時間當作索引，在反轉的雜湊表中找出與索引對應的雜湊值，亦即原雜湊鍵所形成的集合，接著將查詢片段的錨點頻率與頻率差一一拿出來與原雜湊鍵進行比對，如果原雜湊鍵滿足下列兩項條件，則該歌曲編號的分數加一分。

- 原雜湊鍵之錨點頻率 (f1) 與查詢片段內雜湊鍵相同。
- 查詢片段之頻率差 (Δf) 落在原雜湊鍵的 (1-ρ) 與 (1+ρ) 的區間之內，如式 4.2 所示。

$$(1-\rho)\Delta f_{DB} < \Delta f_{Query} < (1+\rho)\Delta f_{DB},\ 0 < \rho \leq 1 \qquad (4.2)$$

doi:10.6342/NTU202000440

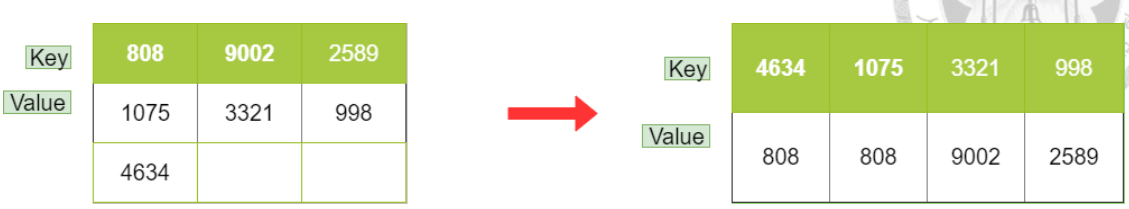

圖 4.11: 反轉雜湊表

將排名前列的歌曲進行重新評分以後，利用式 4.3 將新的分數 ($Score_{rerank}$) 與原始分數 ($Score_{original}$) 結合，得到最終分數 ($Score_{final}$)，最後將最終歌曲排名前十名，與歌曲相關資訊輸出給使用者，流程如圖 4.12 所示。

$$Score_{final} = Score_{original} + \theta \cdot Score_{rerank},\ 0 < \theta \leq 1 \tag{4.3}$$

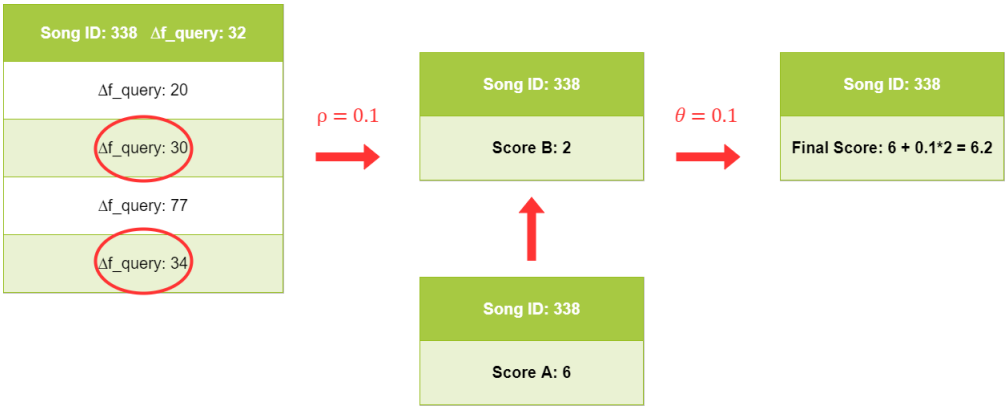

圖 4.12: 重新評分流程示意圖

4.4.3 改進結果與分析

本論文分別採用 $\rho = 0.1, 0.2, 0.3, 0.4, 0.5$ 以及 $\theta = 0.05, 0.1, 0.2, 0.3, 0.4, 0.5$ 進行測試，得到的結果如下圖 4.13、4.14、4.15 以及 4.16 所示。根據單一歌曲

doi:10.6342/NTU202000440

的測試結果，若是將資料庫內 10000 首歌都進行重新評分需要 800 秒，將近 13 分半，因此我們只取原排名前三十的歌曲進行重新評分，所需的平均辨識時間大約落在 800 秒的千分之三左右，也就是 2.5 秒。從測試結果可以看出，在 θ 固定為 0.1 的情形下，ρ 的變動對於辨識時間並無影響，辨識時間都在 2.43 秒與 2.47 秒之間，彼此的差距屬於正常的測試誤差，因為 ρ 僅僅是作為判斷該地標是否計分的條件，並不影響 hit list 的數量，同時也可以看出 ρ 的變動對於辨識率並無顯著影響，辨識率都在 89.17% 至 89.26% 之間跳動，沒有固定的趨勢。接著將焦點放在 θ 上，在固定 ρ 為 0.1 的情形下，θ 等於 0.1 時，我們可以得到較佳的辨識率，CPU 版本為 89.23% ，GPU 版本為 89.3% ，辨識時間也都落在 2.43 秒與 2.47 秒之間，但隨著 θ 超過 0.3 以後，就對辨識率沒有影響了，這是因為當 θ 大於 0.3 時，重新評分的分數 ($Score_{rerank}$) 佔最終分數 ($Score_{final}$) 的比重會大大增加，徹底壓過原始分數 ($Score_{original}$)，這時候的辨識率為 89% ，同時這也是單純使用重新評分的分數當作最終排名時所得到的辨識率，這代表著重新評分的時候反而把錯誤配對的分數拉高了，使得原本最高分的歌曲從正確轉變為錯誤的了，只要頻率差落在區間內便加分的方式還是將部分錯誤的資訊納入了，因此需要利用原始分數作為平衡，利用加權平均重新進行排名以得到較佳的辨識率。圖 4.17 為不同的 ρ 與 θ 組合下之辨識率熱度圖 (heat map) 。

doi:10.6342/NTU202000440

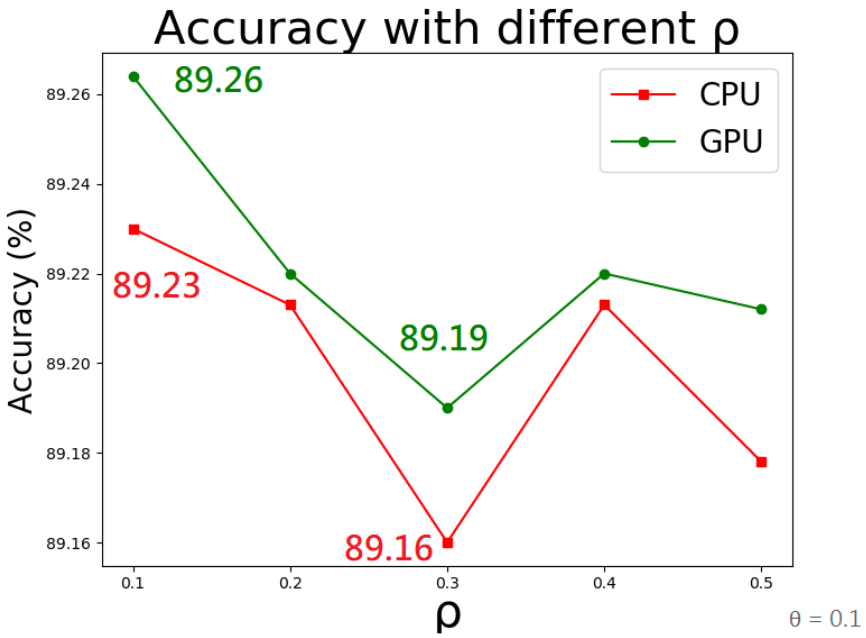

圖 4.13: ρ 與辨識率

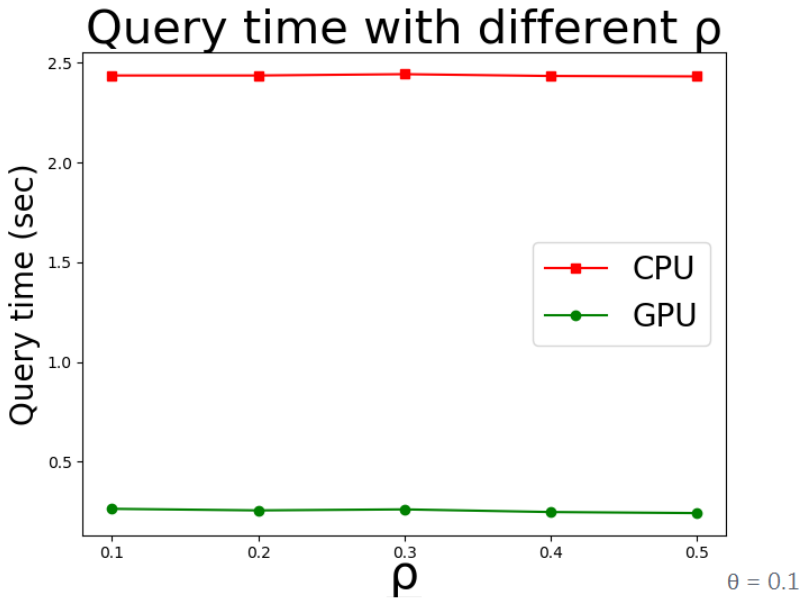

圖 4.14: ρ 與辨識時間

doi:10.6342/NTU202000440

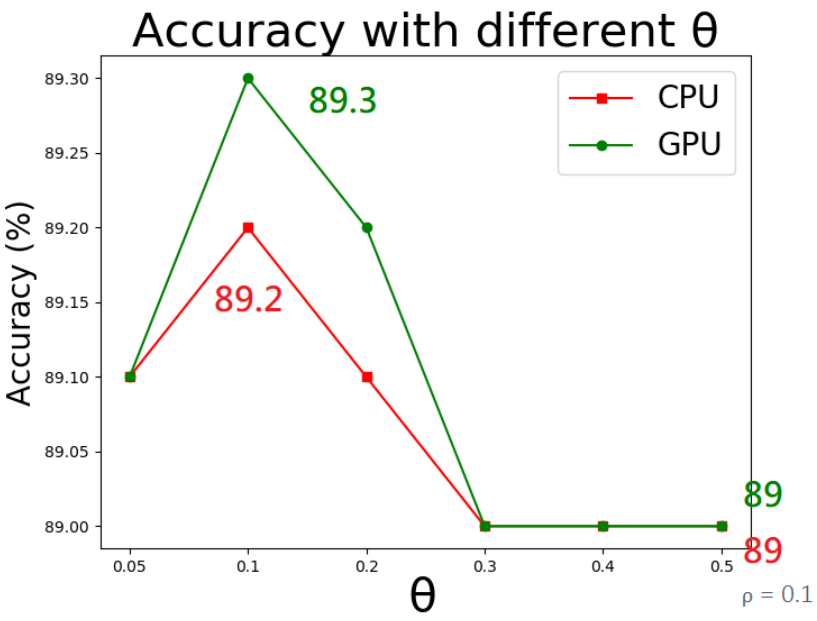

圖 4.15: θ 與辨識率

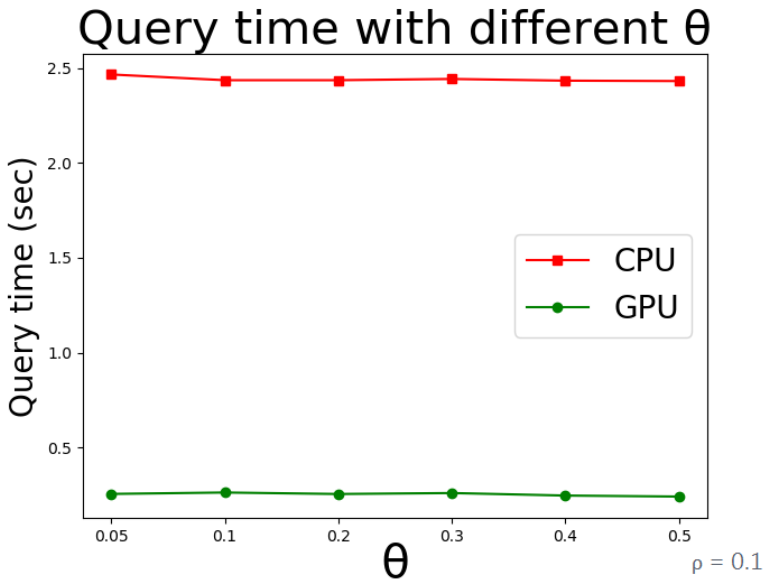

圖 4.16: θ 與辨識時間

doi:10.6342/NTU202000440

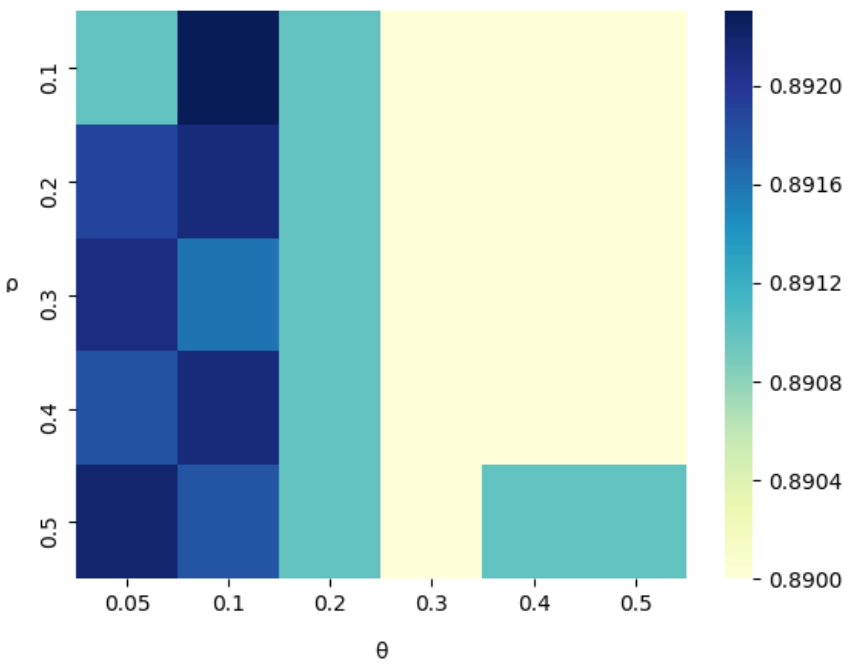

圖 4.17: 辨識率熱度圖

4.5 利用排序學習演算法提升辨識率

4.5.1 改進動機

前面我們利用雙向檢索的方法提升了辨識率，也同時讓每一個查詢片段對資料庫中的每一個歌曲有了相對應的兩種配對評分，為了讓辨識率再往上提升，我們將兩種配對評分當作每一個查詢片段的特徵，利用機器學習方法中的排序學習演算法 (Learning to Rank) 來解析特徵，最後將歌曲重新排名並輸出更為可靠的結果。排序學習演算法的流程如圖 4.17 所示，此演算法最常應用在訊息檢索的領域，使用者給定一個查詢 Q，系統根分級系統給的評分送回一個文件 (document) 序列給使用者，並且該序列中按照文件 d 與 Q 的相關度或重要性依次進行排序。

doi:10.6342/NTU202000440

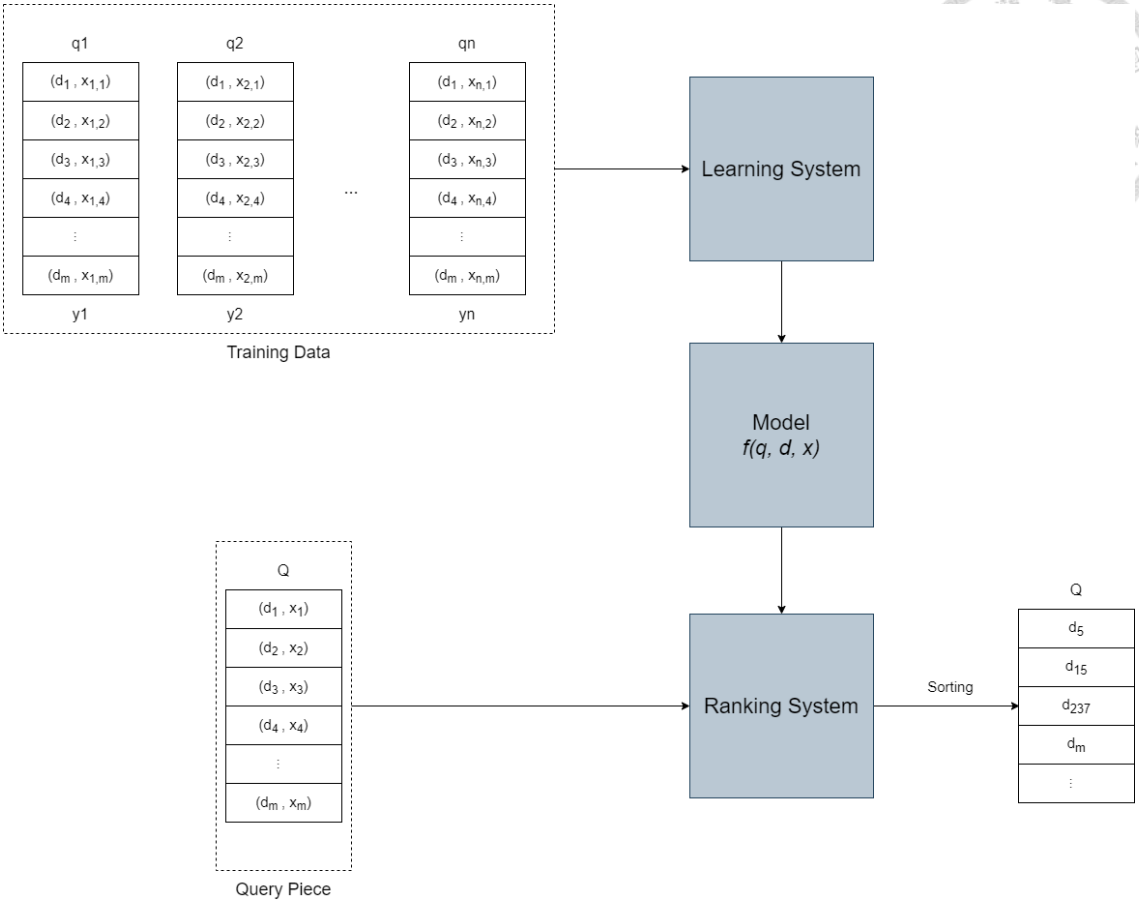

圖 4.18: 排序學習演算法的流程

4.5.2 AdaRank

AdaRank 是由 Jun Xu 於 2007 年發表的一種排序學習演算法 [12]，此演算法是基於 AdaBoost 演算法，更進一步衍伸應用於訊息檢索的領域。首先我們先利用雙向檢索中的兩種配對評分來建構特徵向量，如圖 4.17 所示，訓練資料共有 n 個查詢片段，資料庫中共有 m 首歌曲，每一個查詢片段 (q) 對應到資料庫中的每一首歌曲 (d) 都會有兩種評分，x 代表兩種評分組成的特徵向量，y_k 則代表第 k 個查詢片段的標籤 (label)，是一個大小為 m 的向量，代表此查詢片段與資料庫中每首歌曲的相關性，若是正確配對的歌曲則標記為 1 ，其餘歌曲則標記為 -1 。接下來進行訓練，AdaRank 基本概念是將很多個弱的分級器 (weak ranker) 進行合成變成一個強分級器 (strong ranker) ，我們將訓練資料、迭代次數 T 以及績效評估指

doi:10.6342/NTU202000440

標 E 輸入，這邊使用的是 NDCG (Normalize Discounted Cumulative gain) 來當作評估指標，另外這邊定義排列函數 (permutation function) $\pi(q, d, f)$，輸入查詢片段 q 後藉由分級器 f 我們得到一組歌曲排序 d ，$\pi(j) = j$ 代表排序 d 中的第 j 首歌曲，接著給定每一筆訓練資料的初始權重 $\frac{1}{N}$ ，開始進行迭代，如 Algorithm 2 所示，步驟如下：

- 第一步：藉由訓練資料的權重求得弱分級器
- 第二步：計算這個弱分級器的加權值 α
- 第三步：將目前所有的弱分級器乘上對應的加權值並加總求得強分級器
- 第四步：藉由強分級器回頭更新訓練資料的權重
- 第五步：重複 T 次後輸出最後的強分級器

Data: training data, E, T

Result: strong ranker f_T

Initialize $P_1(i) = \frac{1}{N}$;

for *t = 1 to T* **do**

Create weak ranker h_t with weighted distribution P_t on training data;

Choose $\alpha_t = \frac{1}{2} \cdot ln \frac{\sum_{i=1}^{N} P_t(i)\{1+E(\pi(q_i,d_i,h_t),y_i)\}}{\sum_{i=1}^{N} P_t(i)\{1-E(\pi(q_i,d_i,h_t),y_i)\}}$;

Create $f_t(x) = \sum_{k=1}^{t} \alpha_k \cdot h_k(x)$;

Update $P_{t+1}(i) = \frac{exp\{-E(\pi(q_i,d_i,h_t),y_i)\}}{\sum_{j=1}^{N} exp\{-E(\pi(q_i,d_i,h_t),y_i)\}}$;

end

Output ranking model: $f_T(x)$;

Algorithm 2: AdaRank 演算法

doi:10.6342/NTU202000440

其中弱分級器的求取方法如式 4.4 所示，我們把雙向檢索中所得到的兩種配對評分分別進行排名，把這兩種排名代入評估指標中，看哪一種配對評分帶來的排名可以將評估指標最大化，則該配對評分就是本次的弱分級器。

$$\max_{k} \sum_{i=1}^{N} P_t(i) E(\pi(q_i, d_i, x_k), y_i) \tag{4.4}$$

4.5.3 改進結果與分析

本論文將總共 5692 個查詢片段分成三個部分，前 60% 為訓練集 (training set)，中間 20% 為驗證集 (validation set)，最後 20% 為測試集 (test set)，結果如表 4.5 所示，由於 AdaRank 所需要的特徵向量必須從雙向檢索演算法中獲得，因此我們會先執行雙向檢索演算法獲得特徵向量，接著再輸入訓練完成的排序模型之中重新排序，最後得到的結果，在時間上平均增加約 0.5 秒，辨識率上升 0.8% - 0.9%，有明顯的提升，AdaRank 是序列式方法 (list-wise method) 不僅僅考慮查詢片段與資料庫每首歌曲的相關性，更利用排列函數將歌曲的順序納入學習過程，因此能有效提升辨識率。

	CPU Version	GPU Version
size of database (GB)	3.74	
average query time per song (second)	2.488	0.306
accuracy (%)	90.1%	90.1%

表 4.5: 經過 AdaRank 重新排序後的系統表現

doi:10.6342/NTU202000440

第五章　結論與展望

5.1　總結

本論文提出的改良方法與對應結果如表 5.1 所總結，首先我們藉由調整 GPU 平行運算的演算法，將 CPU 跟 GPU 兩種版本的辨識率拉到幾近一致，並藉由修改計分標準，偏移時間必須完全一樣才計分，使得兩個版本的辨識率都有所提升。接著我們嘗試利用兩張雜湊表，藉由紀錄地標中兩點間的能量關係當作過濾條件，以達到減少 hit list 的長度進而提升辨識速度，更進一步甚至希望辨識率也能有所上升，可惜事與願違，經過分析以後得知此方法易受環境噪音影響，但從實驗結果來看，在 $\rho = 0.5$ 的情況下，我們平均過濾掉了 37% 以上的地標，但同時辨識率僅下降 3.5% ，我們得知若是環境噪音對查詢片段的干擾不大的時候，查詢片段與原曲的能量分布趨勢的確有相同的趨向，而隨著噪音能量越加強大，對查詢片段的干擾也隨之增強，查詢片段與原曲兩者能量分布趨勢的一致性也隨之被破壞。

接著我們針對高能量的噪音所造成的干擾進行了解，發現有許多帶有正確資訊的地標，因為噪音的干擾導致地標中配對突出點的頻率 (f2) 與原值有了差距，使得比對雜湊表時，這些帶有正確資訊的地標無法被納入 hit list ，為了改善這類型的失真，我們反轉雜湊表，並利用雙向檢索將這些地標重新納入評分，並針對原先的分數與新的分數給予不同權重，進行加權平均，得到更高的辨識率。這部分的重點著重於如何在 hit list 中盡可能篩選掉帶有錯誤資訊的地標，並嘗試增加更多帶有正確資訊的地標，以達到減少辨識時間以及提升辨識率。在最後我們引用機器學習的方法針對歌曲的排序進行學習，充份利用系統中兩種不同的配對評分進行學習，更進一步提升辨識能力。

doi:10.6342/NTU202000440

Method	Base line	Alignment of GPU version	Multi-tables with energy	Bi-directional retrieve	AdaRank
Accuracy (%) CPU Ver.	87.2	88.6	85.1	89.23	90.1
Query Time (sec) CPU Ver.	1.14	1.12	0.7051	2.4363	2.488
Accuracy (%) GPU Ver.	85.6	88.7	85.4	89.3	90.1
Query Time (sec) GPU Ver.	0.12	0.15	0.0803	0.26	0.306
Size of database (GB)	3.74	3.74	7.4	3.74	3.74

表 5.1: 各種方法的最佳結果比較

doi:10.6342/NTU202000440

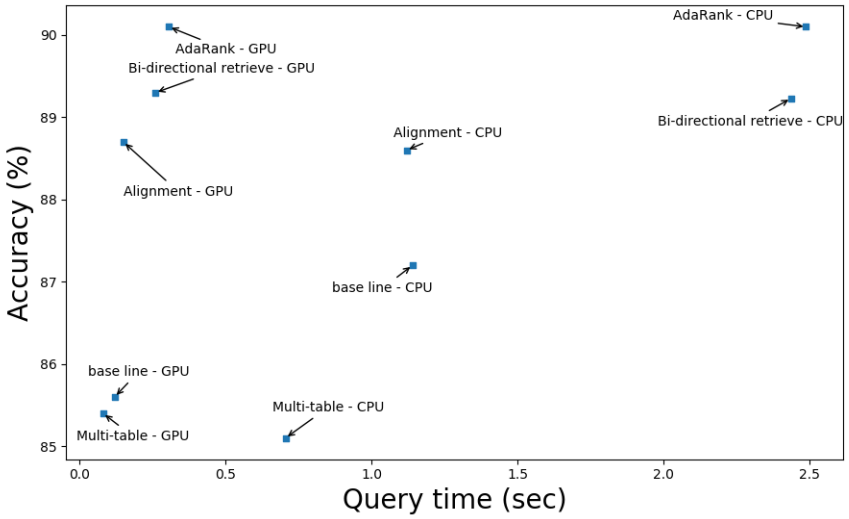

圖 5.1: 各種方法的最佳結果關係圖

5.2 未來展望

本論文一開始成功找到了 GPU 平行運算演算法中造成的誤差並成功拉近 CPU 與 GPU 兩種版本的辨識率差距，但受限於奇偶演算法與執行緒數量的限制，未能完全解決 hit list 中記錄無法全部納入的問題，有鑑於排序演算法在平行運算上面的應用，合併排序 (merge sort) 也許會是完全解決這個問題的方法，未來可以從這個方向入手。而另外一個方向則是盡可能降低雜湊表的所佔據的記憶體空間，現在僅僅是 10000 首歌，雜湊表的大小就有 3.74 GB，我個人想到的可能解決方向有兩種，第一種是對雜湊表進行編碼壓縮，並藉由壓縮過後的雜湊表進行比對，如若辨識率沒有顯著的影響，那麼這一條路或許是可行的；第二種則是建立 SQL 資料庫，將資訊盡可能記錄在硬碟空間之中，雜湊表比對時，只抽出符合條件的雜湊值進入記憶體，有效降低記憶體負擔，雖然這是以降低辨識時間為代價，但我們同時也可能紀錄更多額外資訊進入資料庫而不需擔心記憶體不足的問題，資訊越多，能夠進行的比對就越多種，這一條路很有可能可以同時提升辨識率。最後則是在機器學習的方法上做更多嘗試，序列式方法還有許多種可以應

doi:10.6342/NTU202000440

用在本系統上，而且在訓練集以及測試集的分配上也有許多方法可以嘗試，例如 cross-validation 。以上幾個方向都是有可能進一步提升音訊指紋辨識系統的效能，期望未來能看見更多有趣的方法以改善本系統。

doi:10.6342/NTU202000440

參考文獻

[1] A. L. Wang, "An industrial-strength audio search algorithm," in *ISMIR 2003, 4th Symposium Conference on Music Information Retrieval*, 2003, pp. 7–13.

[2] Hyoung-Gook Kim, Hye-Seung Cho, and Jin Young Kim, "Robust audio fingerprinting using peak-pair-based hash of non-repeating foreground audio in a real environment," *Cluster Computing*, vol. 19, pp. 315–323, 2016.

[3] Hsin-Fu Liao, "Improvement of landmark-based audio fingerprinting with target zone and hash table tuning," M.S. thesis, Department of Computer Science and Information Engineering, College of Electrical Engineering and Computer Science, National Taiwan University, 2018.

[4] "音 樂 史," https://zh.wikipedia.org/wiki/%E9%9F%B3%E6%A8%82%E5%8F%B2, Accessed: 2019-11-12.

[5] "Soundhound," https://soundhound.com/, Accessed: 2019-08-30.

[6] "Shazam," https://www.shazam.com/, Accessed: 2019-08-30.

[7] "Echonest," http://the.echonest.com/, Accessed: 2019-08-30.

[8] Chris J.C. Burges, Dan Plastina, John Platt, Erin Renshaw, and Rico Malvar, "Using audio fingerprinting for duplicate detection and thumbnail generation," https://www.microsoft.com/en-us/research/publication/using-audio-fingerprinting-for-duplicate-detection-and-thumbnail-generation/, Accessed: 2019-10-21.

doi:10.6342/NTU202000440

[9] Dan Ellis, "Robust landmark-based audio fingerprinting," http://labrosa.ee.columbia.edu/matlab/fingerprint/, 2009, Accessed: 2019-08-30.

[10] Jijun Deng, Wanggen Wan, Ram Swaminathan, Xiaoqing Yu, and Xueqian Pan, "An audio fingerprinting system based on spectral energy structure," in *IET International Conference on Smart and Sustainable City (ICSSC 2011)*, 2011.

[11] Xueqian Pan, Xiaoqing Yu, Jijun Deng, Wei Yang, and Hongxue Wang, "Audio fingerprinting based on local energy centroid," in *IET International Communication Conference on Wireless Mobile and Computing (CCWMC 2011)*, 2011.

[12] Jun Xu and Hang Li, "AdaRank: A Boosting Algorithm for Information Retrieval," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2007)*, 2007.

doi:10.6342/NTU202000440