A Study of Using Algorithmic Composition Based Analysis with Auto Accompaniment to Synthesize music Automatically: Case study in Classical Jazz

以演算法作曲爲基礎之分析於自動化伴奏與音樂合成研究：古典爵士樂案例分析

中文摘要：

本論文以演算法作曲 (algorithmic composition) 爲基礎，使用馬克夫鏈 (Markov chain) 先產生伴奏和聲的部分，再爲其譜上旋律，並使用決策樹 (Decision tree) 爲其增添即興部分。

系統以爵士樂爲主，分爲兩大部分，第一爲解構樂曲，分析和聲、音程、以及音高的部分：利用樂曲音符的出現率，以及和聲分析的所有法則，由此加以判斷樂曲之和聲進行走向，與旋律的音高排列方式。第二則是利用上述的分析系統，分析樂曲，並使用馬克夫鏈而產生自動伴奏，再爲其加入旋律與節奏的進行規則，由此而產生完整的爵士樂曲 。

本論文提供問卷讓受訪者評論，問卷結果顯示專業的爵士樂手評價最高，其次爲非音樂專長者，而古典樂背景則接受度較低。

關鍵字：

電腦音樂、自動伴奏、馬克夫鏈、MIDI、自動作曲、演算法作曲。

Abstract：

Algorithmic composition, a type of automated music composition system invented among composers in which specific set rules are designed and its calculations are carried out by a computer. This thesis is based on algorithmic composition, to compose the part of harmony of accompanying with markov chain, and the part of improvisation with decision tree.

The system takes jazz as the main subject, and divides it into two major parts, including auto-analysis, which analyzes harmony, interval and pitch class, and the second part auto-composition which synthesize music elements.

This thesis offers the questionnaires to let interviewees comment on, and reveals that Jazz musicians make the highest point of comment. The second type of the comment is from the non musician, however, the classical musician make the lowest point of comment .

Key words:

Computer music、Auto-composition、Markov chain、MIDI、Auto composition、Algorithmic composition、Decision tree。

誌謝

研究所三年的時間，在過的時候，覺得很漫長，但是經過以後，卻又覺得快得令人不可思議，說起來，這真的是人生中最充實，也最奇妙的三年了！從小音樂班到大學的我，從來沒有想過會一頭栽進這個零與一，充滿邏輯判斷與數學式的電腦領域中，更沒有想過有一天會參加軟體設計比賽，會用程式來當畢業論文。

經歷了這一切，當然，最需要感謝的人，就是我的指導老師黃志方教授，真的很難找出一個適當的詞彙來表達這三年來對您的感謝，從一入學至今，總是在您慈祥又嚴格的督促中學習，總是讓您在百忙中抽出時間來幫助我們，感謝您在這三年來，對我的教導，能夠成爲您的學生，真的是很幸運的一件事！

感謝Phil Winsor老師，感謝您在電腦程式作曲中，給予我們的教導，在您身上，真的獲益良多。

此外，也感謝在竹韻中，對我悉心教導的董老師，在那一段跟您學習作曲的日子裡，真的非常充實，也深深體會到創作的快樂！

感謝科技組裡的每一位陪伴我的同學及學長，幸輯學長、擎亞、珍妮、以及士涵、奕佐，這三年來，大家一起度過了好多艱難與快樂的時光，不管是趕作業的苦海，畢業的壓力，或是每一次的聚餐與玩樂，都很高興有你們在一起！

感謝我最親愛的兩位室友，舜評以及琦惠，同住的這兩年，真的是悲喜與共，很開心有你們兩位最可愛的室友，分擔所有生活的壓力與快樂！

最後，就是要感謝我親愛的家人，謝謝你們的支持，也謝謝一路上你們對我的照顧與幫助！

目錄

圖目錄

表目錄

第一章、緒論:

第一節 動機

隨著電腦科技的進步，電腦輔助作曲軟體、音序器 (Sequencer) 、與各種改變聲音聲響的音色庫、效果器等逐漸開發成熟，進而，使用電腦來代替創作以及編曲，甚至配伴奏的演算法作曲技術，也迅速發展。

使用電腦軟體來製作音樂，對大眾普遍人而言，並不是一門簡單的學問，首先要有音樂的概念，以及創作的靈感與巧思，而對完全沒有涉略過樂理或和聲基礎概念的人，這些都是相當困難的。

然而，透過科技的發展，本系統便是希望可以經由電腦資訊，將音樂的創作方法加以程式化，包括和聲進行、音程起伏、還有許多作曲語法，轉變爲沒有音樂基礎的人都可以上手的簡單工具，本系統透過強大的讀寫功能，希望藉由輸入的作品曲風及樂曲參數，模擬出更多的風格寫作音樂。

演算法作曲的技術，在學術上的發展已有相當久遠的歷史；本論文除了利用演算法來創作樂曲以外，由於樂曲的分析，對於許多學者，甚至是音樂專業人來說，都是一門相當耗時的學問，本論文除了樂曲產生系統，更額外的增添了分析系統，透過音樂語法的演算，以及機率統計，藉由電腦大量而快速的運算，將樂曲的元素，轉變爲可以量化的參數。

透過這樣的分析，不但可以爲樂曲產生系統，提供大量的樂曲資訊，更讓系統成爲一個開放的空間，經由輸入新的風格之樂曲，從而產生更多樂種的風格模組。除此，對於音樂學者或分析者而言，這樣用以解構樂曲的系統，更可以達到

樂曲大量而快速的分析。

爵士音樂在眾多流行音樂之中，是在世界上影響最廣的一個樂種；同時也是美國最有代表性的民間音樂。對普遍大眾而言，爵士樂可以是怡情養性的流行音樂，而對於學術研究而言，爵士樂的悠久發展歷史，以及特殊的和聲及作曲手法，都相當具有研究的價值。

本論文以商業及學術之間，都相當具有發展價值的爵士樂，作爲分析的樂種，希望藉由本篇的分析與創作系統，爲電腦音樂的創作，開發一個簡易並大眾化的分析與作曲軟體。

第二節 背景介紹

演算法作曲，是試圖使用某個形式化的過程，可使創作者在利用電腦創作樂曲時，達到最小的介入程度之研究。而自動伴奏系統，多半是利用基本的樂理與統計方法作爲理論基礎，以執行和絃的辯識，並爲主旋律譜上伴奏。這些技術的發展，使創作者可以在不了解樂理或音樂概念時，也能夠輕鬆利用電腦輔助，進行音樂的創作。

有關音樂的形式化，最早可以追溯到作曲家爲複調音樂發展的嚴格規則，包括主題動機實施倒影、主題延長、縮減這樣系統化的過程來演繹。音樂創作的形式化在二十世紀被再一次的使用，而探索及研究演算法作曲的技術，除了實用的價值外，也可以讓我們瞭解和類比作曲家在創作音樂過程中的思維方式，甚至用邏輯化的方法，對作曲家的作品進行分析。

以下簡介國內外關於演算法作曲、自動伴奏等相關研究：

一、自動伴奏系統 :

自動化伴奏系統 [Onishi, Niizeki, & Kimura, 2001] 包含利用基礎樂理與統計方法作爲基礎來執行和絃辯識，並將此兩種方法的分析結果，利用 Dynamic programming 演算法找出最佳的和絃解答。讓使用者不需利用鍵盤輸入，也不需有樂理基礎就可以利用辨識出的和絃結果爲旋律配上伴奏。

二、使用馬克夫鏈轉換表的演算法作曲方法 [Orio & Déchelle, 2001] [Rabiner, 1989] [Schwarz, Orio, & Schnell, 2004] :

在演算法作曲中，一個簡單但有趣的技術是按照一個轉換表來依次選擇音符。這個轉換表就像一個函數，其引數是當前的音符，而函數值則是下一個要出現音符的可能性。轉換表可以按照一定的標準手工構造，並且嵌套一個特定的音樂風格。針對某一特定(如某一作曲家或某一時期)風格的音樂作品(樣板集合)進行收集和統計，就可以構造出相應的轉換表。而這個轉換表定義了這些特定音樂風格的作品(樣板集合)中音符導向的可能性。

Cybernetic Composer系統就是這類模型的一個範例 [Ames & Domino, 1992] 。該系統可以創作出諸如爵士樂、搖擺樂及繁拍子 (Ragtime) 音樂這些不同風格的音樂片斷。這一系統的一個有趣特點是：它先用馬克夫鏈推演出旋律的節奏 (前一節的馬克夫轉換表中關於旋律的音高推演，亦可平行地用在節奏的推演上)，而在較後的階段再推演音高。

三、基於規則的知識庫系統：

音樂知識庫的使用，似乎是很自然的一種選擇。特別是當我們試圖在已定義完善的領域內建立模型或者是介紹一個清晰的結構或規則時尤其如此。其主要優

點是：它們具有清晰的推理，並能夠爲行爲的選擇作出解釋。Ebciogln專門建立了一種回溯說明語言 (backtracking specification language，簡稱BSL) ，並且用它來實現CHORAL——一個基於規則的專家系統 [Ebciogln, 1986] 。它可以構造具有巴赫風格的四聲部合唱曲。該系統包含大約350條規則，它們以一階謂詞的演算形式書寫 (即這裡的回溯說明語言) 。這些規則從合唱曲的多個角度來描述音樂知識，例如，和絃骨架以及多個單聲部的旋律線等。

四、自動作曲的音樂文法：

正如語言有文法一樣，音樂也是有音樂文法的。Steedman發明了一種生成文法 [Steedman, 1984] ，用以描述爵士樂12小節藍調的和絃進程。後來他又使用範疇文法，對其原有工作做進一步的精煉。他的系統允許對傳統上被視爲右分叉的結構作左分叉分析，從而提高了系統從左到右的解釋能力。這使得系統能夠類比出更接近聽者的直覺，且所產生的和絃演繹進程更容易讓人接受。

五、開放式的爵士樂自動伴奏演算－提供使用者可以合作開發的人工智慧系統：

大部分的演算法作曲系統，都是用讓開發者使用樂理知識，定義功能與資料庫，而爲了符合必要的音樂特質，Beer-Sheva開發一個開放式的系統 [Sheva, 1996] ，可以創作用鼓、Bass、以及鋼琴組成的伴奏系統，將音樂知識從實際的伴奏演算法分割，從使用者的操作中，從而吸收成長，提供一個相當於普通人可與爵士專家共同合作成長的系統。

六、使用正規語法產生爵士和聲序列的即興創作 [Chemillier, 2000] :

以爵士和絃的基本架構爲原理而產生和聲序列，並加入作曲法的資料庫，從

樂曲的採樣中產生和聲。由於爵士樂和絃的伴奏法，通常爲N小節的和聲不停反覆並且加入即興變奏，鋼琴演奏者會即興式的使用新和聲來代替原本的和絃，這樣的研究，可以達到此種即興式的改變基本和絃，並加入替代和絃的技巧。

本系統參考上述的幾種方法，將馬克夫鏈運用在和聲的進行當中，藉由轉換表來決定下一個出現的和聲機率，使用自動作曲的音樂語法，產生即興部分的和聲外音，並建立節奏的資料庫，藉以產生豐富的爵士樂曲準則創作。

而除了參考過去許多人所使用過的準則方法以外，本系統更額外增添了分析系統，將樂理知識運用在程式上，結合快速的電腦運算，代替研究者完成大量的分析，讓準則作曲不再只是作曲的功能，更可以代替人類去完成樂曲分析的工作也讓系統，從被動的需要程式創作者寫入規則，成爲一個可以主動自行吸收並增加資料庫的人工智慧。

第二章、研究方法：

第一節 系統簡介

本論文以爵士樂曲爲主，針對爵士音樂語法中的許多規則性，如和聲、聲部對位方式等，利用大量的取樣分析，以及作曲家樂理規則的套用，類比音樂家在創作時的思維方式，先產生自動伴奏部分的和聲，再依照伴奏使用的和聲，套用某些樂理或旋律的進行方式與規則，將主旋律也一起產生，進而創作出完整的樂曲。

本系統分爲兩大部分，其一爲解構樂曲的分析部分，將讀入的 Midi 檔案進行分析，列出樂曲中每一個和聲的進行，以及音程與音高的出現率。和聲的架構，是產生自動伴奏最重要的部分；而音程與音高，在旋律產生系統中，兩者都是相當重要的元素；其二爲自動伴奏與旋律產生系統，運用馬克夫鏈來產生類比爵士樂作曲家的音樂寫作模式，模擬出伴奏部分的和聲進行走向，配合音高與音程的採樣，再用爵士樂的樂理規則爲旋律線條加上爵士風味的和聲外音，由此而產生完整的爵士樂曲。

以下系統大綱圖:

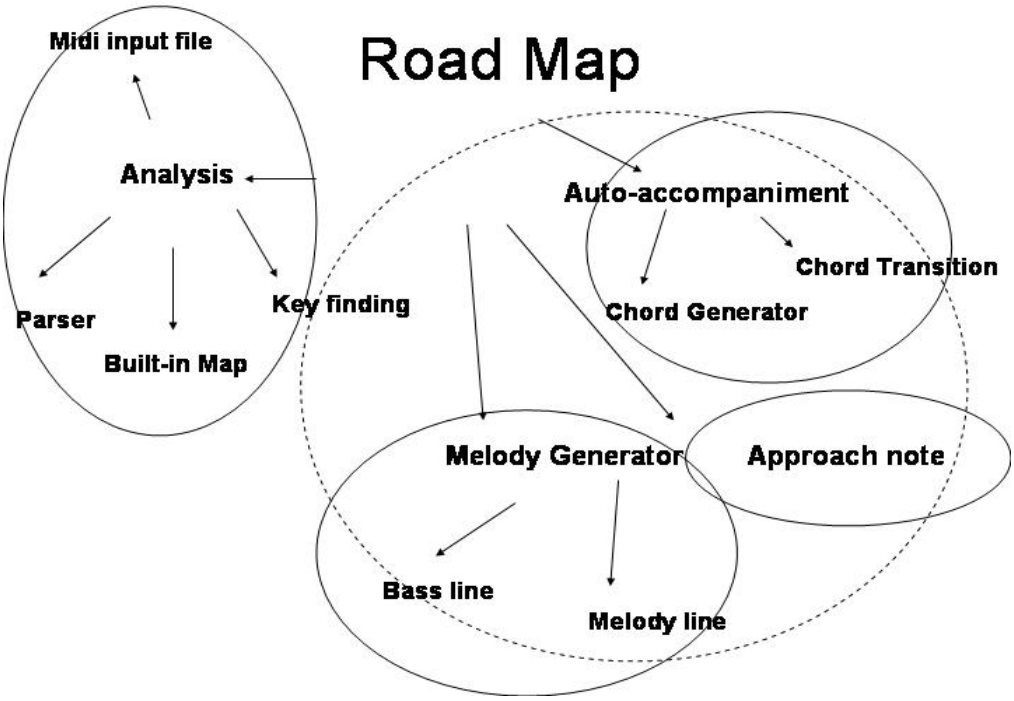


圖 2-1 系統大綱圖

以下爲系統流程圖：

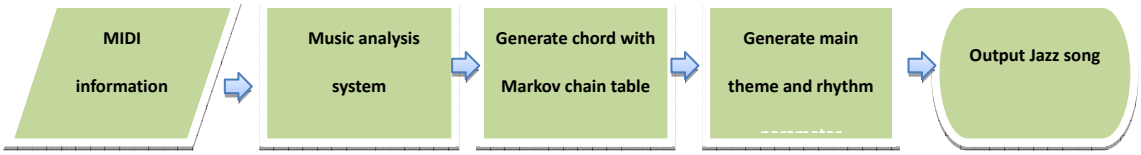


圖 2-2 系統流程圖

第二節 樂曲分析系統:

本分析系統輸入資訊以爵士樂 Midi 資料爲主，爲樂曲進行音程度數分析，和聲進行分析，以及各和絃內的音類 (pitch class) [Straus, 2009] 分析，以下爲分析系統大綱圖：

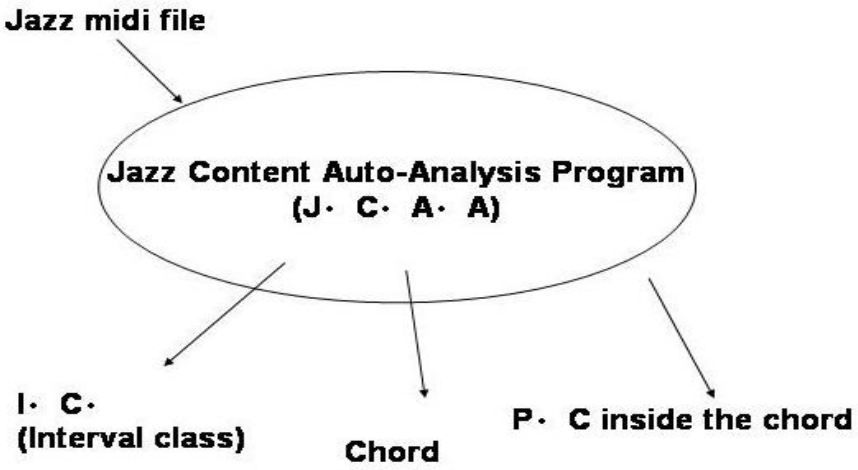


圖 2-3 分析系統大綱圖

一、音程分析與音類

在西方的和聲學理論中，以巴哈的十二平均率爲基礎，音程就是兩個音的高度關係，計算音程最小的單位爲半音，下面爲 C 大調音階中的音程表：

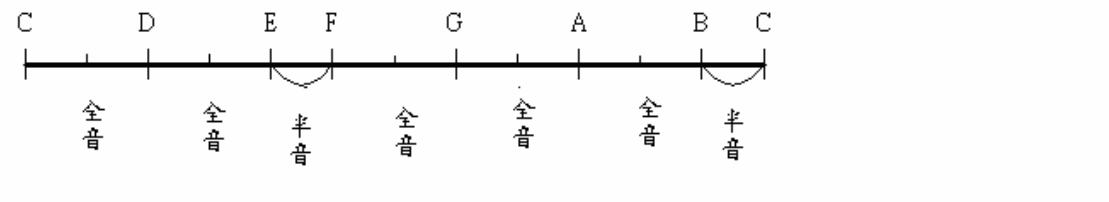


表 2-1 音程

音程大於一個八度稱爲複音程 (Compound Interval) ；小於一個八度則稱爲單音程 (Single Interval) 。而本論文所分析的音程全部都是使用單音程來紀錄，因此大於八度的音程，音級數目都要減去七度，例如完全十一度音程，則視爲完全四度音程，因爲複音程的特性是以單音程的特性爲基礎，所以這兩個音程的特性也是相同的。

音程的名稱除了兩個音級相對的數字以外，還包括下列這些詞彙:大、小、增、減、完全，這些都是用來詳述音程的特性。

下列爲半音數量與音程名稱的對照表:

半音數量	音程名稱	等音異名 (Enharmonic) 音程
0	完全一度 (P1)	減二度(dim2)
1	小二度（m2）	增一度(aug1)
2	大二度(M2)	減三度（dim3）
3	小三度(m4)	增二度(aug2)
4	大三度(M3)	減四度(dim4)

5	完全四度(P4)	增三度(aug3)
6	增四度(aug4)/減五度(dim5)	減五度(dim5)/ 增四度(aug4)
7	完全五度(P5)	減六度(dim6)
8	小六度(m6)	增五度(aug5)
9	大六度(M6)	減七度(din7)
10	小七度(m7)	增六度(aug6)
11	大七度(M7)	減八度(dim8)
12	完全八度(P8)	增七度(aug7)

表 2-2 半音數量與音程對照

音類在音樂上意指各個八度內十二音列中的音，在音類內 C 音代表每個八度的 C，用科學上的符號來代表 C 的音類，可用下列的公式表達:

$$\{C_n : n \text{ is an integer}\} = \{..., C_{-2}, C_{-1}, C_0, C_1, C_2, C_3 ...\} \quad (1)$$

而爲了避免同音異名的現象，有一種以音高頻率來表達的方程式爲:

$$p = 69 + 12\log_2(f / 440) \quad (2)$$

在 Midi 的編碼系統內，使用 0 到 127 來代表所有的音高，要明確的表達 pitch class set，就要清楚的分類屬於同一類集的音名，也就是以 12 爲商數的餘數 X，$0 \le x < 12$。因此，使用 0=C，1=C#來表達。

二、和聲進行分析

演算法作曲之研究，許多都是要利用大量的作曲素材加以分析，其中，"和聲"就是作曲中相當重要的一大元素。甚至樂曲的色彩與張力，很大一部分，都是取決於和聲的進行走向。然而，分析和聲，卻是一門曠日廢時的工作，若是取樣的作品到達相當數量，則將此分析和聲之工作加以程式化，便是相當需要的。

由於流行音樂中關係大小調和絃共用的現象十分普遍，如果用經典和聲學解釋，經常會有混淆的狀況。因此本系統採用流行音樂的樂理記譜法，不考慮調性，將樂曲中的和絃以絕對音名的記譜法輸出，格式如同一般大眾接觸的吉他、以及流行樂譜的記譜方式。

本系統分析以七和絃爲主，然而，高數目和聲，如九和絃、十三和絃等，卻是構成爵士樂相當重要的部分，而功能上卻是與三和絃、七和絃相當。因此在分析系統產生的三和絃、七和絃，我們視高數目和絃爲三、七和絃加上一些「贅音」(Redundant notes) ，於產生系統的部分，則會用以其它高數目和絃代替，進而產生更符合爵士風格的樂曲。

以下做簡單的爵士樂和聲記法介紹：

1 大三和絃：根音與三音是大三度，三音與五音是小三度，用根音的大寫英文字母音名來表示，如 DO，MI，SOL 和絃用 CM 表示。

2 小三和絃：根音與三音是小三度，三音與五音是大三度，用根音的大寫英文字母音名加上小寫 m 表示，如 RE，FA，LA 和絃用 Dm 表示。

3 增三和絃：根音與三音，三音與五音都是大三度，用根音的大寫英文字母音名加一個"+"。如 DO，MI，升 SOL 和絃表示爲 C+。

4 減三和絃：根音與三音，三音與五音都是小三度，用根音的大寫英文字母音名加上 dim 或一個 "-"。如 RE，FA，降 LA，表示爲 D-。

5 屬七和絃：在大三和絃基礎上再加小三度，用根音的大寫英文字母音名加上"7"即可，如 SOL，SI，RE，FA 和絃用 G7 表示。

6 大七和絃：在大三和絃基礎上再加大三度，用根音的大寫英文字母音名加上 maj7 表示，如 DO，MI，SOL，SI 和絃表示爲 Cmaj7，降 SI，RE，FA，LA 和絃表示爲 Bbmaj7。

7 小七和絃：在小三和絃基礎上再加小三度，用根音的大寫英文字母音名加上"m7"表示。如 LA，DO，MI，SOL 和絃表示爲 Am7，RE，FA，LA，DO 和絃表示爲 Dm7。

8 小大七和絃：在小三和絃基礎上再加大三度，用根音的大寫英文字母音名加上 mM7 表示，如 DO，降 MI，SOL，SI 和絃表示爲 CmM7，LA，DO，MI，升 SOL 表示爲 AmM7。

9 減七和絃：在減三和絃的基礎上再加小三度，用根音的大寫英文字母音名加上 dim7 表示，如 SI，RE，FA，降 LA 和絃表示爲 Bdim7，LA，DO，降 MI，降 SOL 表示爲 Adim7。

10 半減七和絃：在減三和絃的基礎上再加大三度，用根音的大寫英文字母音名

加上 m7-5 表示，如 SI，RE，FA，LA 和絃表示爲 Bm7-5，升 FA，LA，DO，ME 和絃表示爲 F#m7-5。

11 增七和絃：在增三和絃的基礎上再加小三度，用根音的大寫英文字母音名加上+7 表示，如 DO，MI，升 SOL，SI 和絃表示爲 C+7。

第三節 爵士樂自動伴奏與旋律產生系統

本樂曲產生系統，將上述分析系統所得到的參數，經由各種演算法運算，重新產生輸入樂曲之風格寫作，以下爲系統大綱圖:

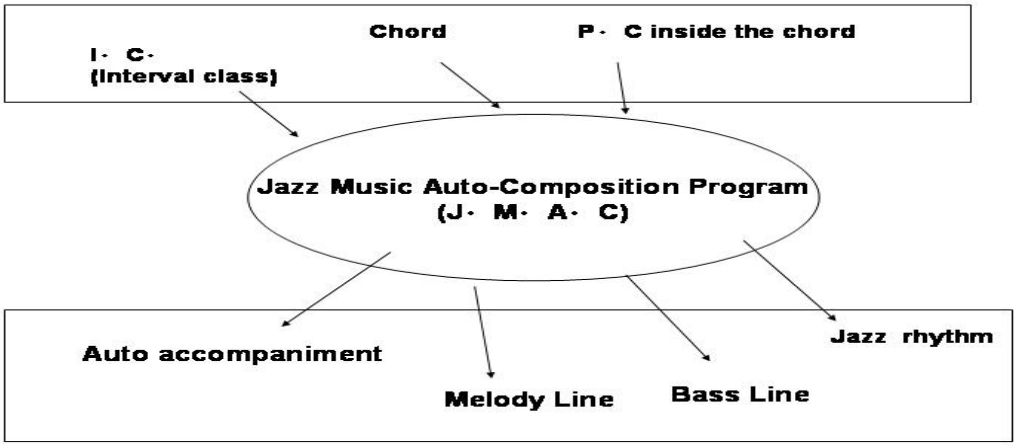


圖 2-4 產生系統大綱圖

一、爵士樂及分析作品簡介

爵士樂在眾多流行音樂之中，是在世界上影響最廣的一個樂種；同時也是美國最有代表性的民間音樂。爵士樂起源在 17-18 世紀，大批非洲黑人販被賣到美

洲，常常唱一種哀歌，以表達他們痛苦的心聲。這種音樂也滲透在一些宗教 歌曲中。由於藍色在美國人民中，被看作是憂鬱的色彩，這種悲哀的歌曲，也就統稱之爲 Blues (藍調) 。早期的黑人爵士樂師多不識譜，演奏時只是憑著靈感，對所熟悉的曲調自由地進行變化。所以帶有很強的即興性質。

爵士樂隨著歷史的發展，曲風相當多元且豐富，包括 Big band swing、Bebop、Cool、Hardbop、Postbop、Free jazz、 Fusion、 Jazz blues、 Ragtime 等。而本論文則以 Blues、 Ragtime 的分析及創作爲主。

Blues 是一種基於五聲音階的聲樂和樂器音樂，它的另一個特點是其特殊的和聲。藍調中使用的「藍調之音」後來對於美國和西方的流行音樂有非常大的影響，Ragtime 也是從中發展而來。兩者在和聲及音階的運用上，都有極大的相似度，有許多作曲家，將 Blues 與 Ragtime 成功結合，作曲家 W·C·Handy 便是其中相當傑出的一位。

1930年代12節藍調成爲標準。但是除典型的12節藍調外還有8節藍調形式和16節藍調形式。基本的12節藍調反應了標準的和聲，其節奏爲4/4或2/4拍。

以下爲C調藍調基本的和聲架構，下圖由左至右四小節爲一段落，順序由上而下：

C	C	C	C7
F	F	C	C
G	G7	C	C

表 2-3 爵士樂和聲架構

上述的和聲爲初期較工整簡單的和聲架構，提出僅供參考，本論文分析的和聲結果，並不按照上述單純的結構所言，而是更複雜而多元的和聲進行。

本論文分析內容包括 W·C·Handy、Scott Joplin 等爵士樂大師的作品，以下簡單介紹這些作曲家的樂風及特色：

W·C·Handy 是第一位將藍調改寫給交響樂團的人，最著名的作品爲聖路易藍調，他的作品可以看作是 Blues、Ragtime 的綜合。

Scott Joplin 在 Ragtime 音樂上的卓越成，使他贏得了「雷格之王」的雅號，爲全盛時期的 Ragtime 音樂注入了清流，他多樣的創作，使得 Ragtime 有了清新、優雅的風格。Joplin 的音樂可說是正統 Ragtime 的經典，最著明的有 (Maple Leaf Rag)、 (The Entertainer) 等曲。

二、圖形化介面

由以上的和聲分析系統，對大量的爵士樂曲進行和聲分析之後，將十分容易的得到樂曲和聲資訊。本論文即是透過對和聲的採樣分析，對照馬克夫鏈的換算表，重新產生類比原先樂曲的和聲進行模式，先產生自動伴奏部分的和聲進行，並依據音類與音程的採樣，爲和聲填入旋律，並以演算法樹的判斷方式，爲旋律加入爵士即興的部分，藉以創作出更符合爵士風味的主旋律線條，最後配合節奏參數，由此而生成完整的爵士樂曲。

本系統設計爲簡易上手的圖形化使用介面，在產生樂曲後，可以任意移動全曲的音高，以及速度。下圖爲本系統的圖形化介面:

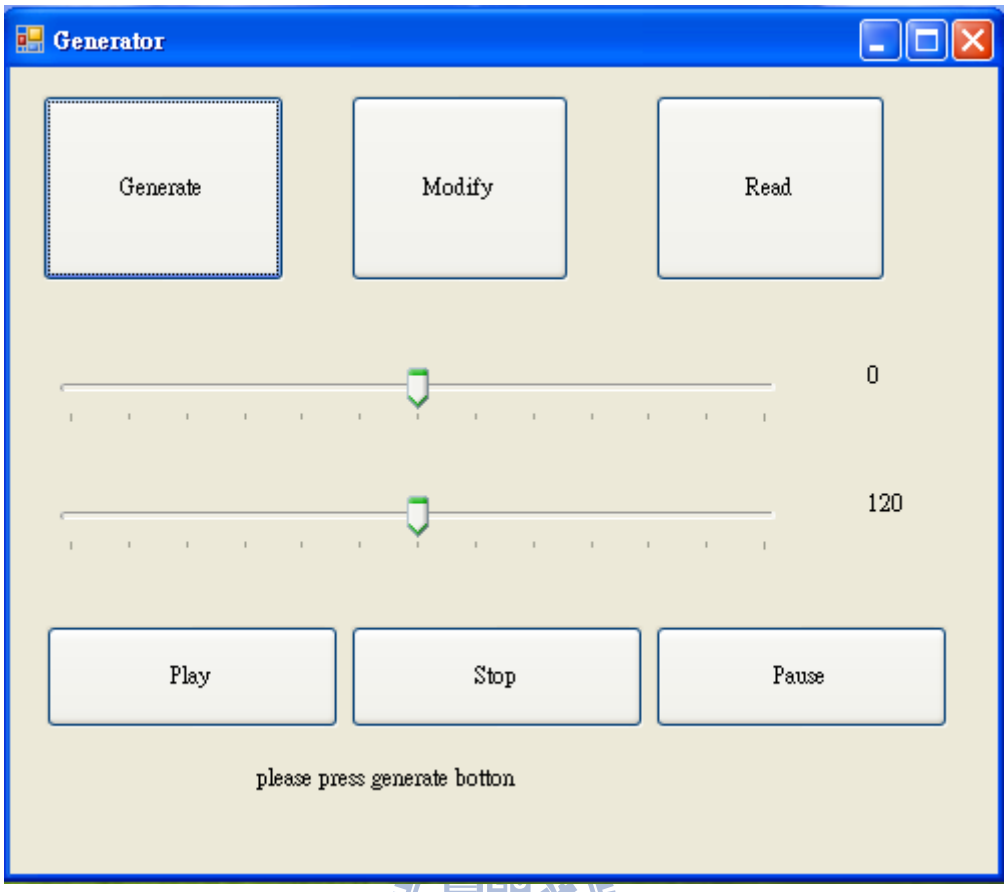


圖 2-5 圖形化介面

下圖爲操作方式流程圖：

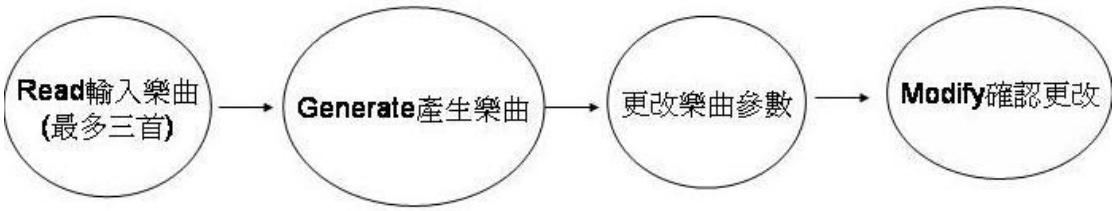


圖 2-6 操作流程圖

第三章、實現方法：

本論文以流行爵士樂的分析爲主，分析內容包括W·C·Handy、Scott Joplin爵士樂大師的作品。參考David Temperley 所著作之 Music and probability [Temperley, 2007] 判斷樂曲調性，依調性將樂曲加以分類，而判斷時，則使用爵士樂的和絃記譜法做判斷，考慮的範圍也限制在三和絃與七和絃，九和絃以上，都不在考慮範圍內。

本系統的操作方法，是將Midi檔案直接輸入，經過電腦運算後，再輸出和絃名稱，音程分佈，以及各和絃中包含的音類。經由分析大量樂曲之後，樂曲產生系統將會利用馬克夫鏈產生和聲進行，並使用分析結果數據，來填上和聲中相對應的音高及音程，最後利用演算法樹的判斷方式，加入即興手法，並配合節奏參數，由此產生完整的爵士樂曲。

以下爲實現方法流程圖:

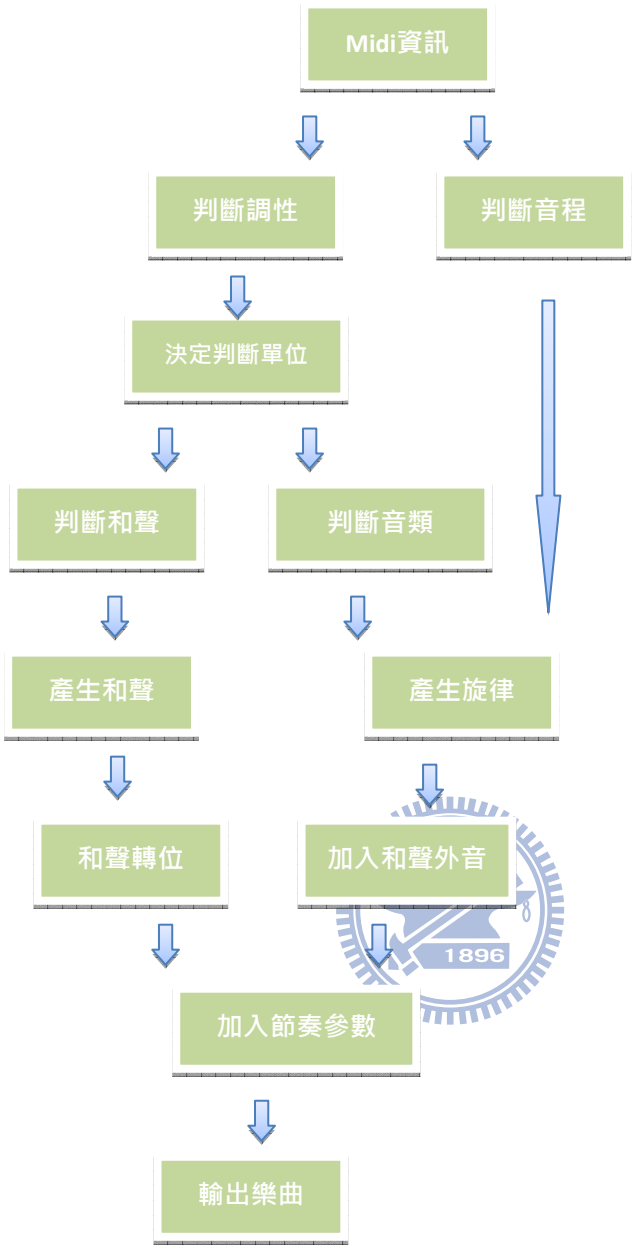

Midi資訊
判斷調性
判斷音程
決定判斷單位
判斷和聲
判斷音類
產生和聲
產生旋律
和聲轉位
加入和聲外音
加入節奏參數
輸出樂曲

圖 3-1 實現方法流程圖

第一節 樂曲分析方法

本節講述樂曲分析系統的分析方式，輸入資訊爲爵士樂的 Midi 檔案，輸出資訊包括音程、調性、和聲與各和聲內音類。

一、簡介 Midi 技術的編碼方式

在 MIDI 資料內，以中央的八度爲例，任一半音都相差一個數字。以下以中央 Do 的八度音爲例，數字代表 Midi Note Name 資訊的編碼：

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
60	61	62	63	64	65	66	67	68	69	70	71

表 3-1 Midi 資訊編碼

由於判斷和絃與音程，任一八度的音高，都不會影響判斷結果，爲了方便資料的查詢與比對，使用音類，將所有的音高數字都 Modulus 12。

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	1	2	3	4	5	6	7	8	9	10	11

表 3-2 音類

二、音程分析方法

音程，是樂曲旋律線條中，決定音高相對關係的重要參數之一，音程的判斷與調性沒有直接關係，爲分析系統內較爲單純的一種。全部的輸出資料都是以單音程爲主。

以下爲兩音之間的半音數量與本分析系統輸出格式的對照表:

半音數量	0	1	2	3	4	5	6	7	8	9	10	11
音程名稱	P1	m2	M2	m3	M3	P4	aug4	P5	m6	M6	m7	M7

表 3-3 音程

以下爲音程判斷流程圖:

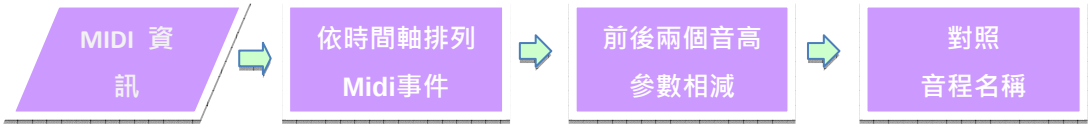


圖 3-2 音程判斷流程圖

三、調性判斷與和絃判斷流程

在判斷樂曲的和絃前，系統先將樂曲依照調性分類，以避免在統計時，造成調性的混淆。本論文分析調性的方法則參考 David Temperley 所著作之 Music and Probability。其判斷公式爲：

$$P(\text{pitch sequence} \cap k \cap c) = P(\text{pitch sequence} \mid k \cap c)P(k \cap c)$$

$$= P(k)P(c)\prod_{n} RPK_n \qquad \textbf{(3)}$$

在公式中c＝central pitch；k=key profiles；RPK=product of key, range, and proximity profiles；n=integer。

判斷結束之後，依調性將樂曲加以分類，而判斷和絃時，則使用絕對音高的和絃做判斷，考慮的範圍也限制在三和絃與七和絃，九和絃以上，都不在考慮範圍內。

以下爲和絃與音高對照圖 (Chord match) 只泛指出白鍵上的和聲:

大三和絃	CM={0,4 ,7 }	DM={2,6 ,9 }	EM={4, 8 , 11 }
小三和絃	Cm={0,3 ,7 }	Dm={2, 5, 9 }	Em={4 , 7 , 11 }
增三和絃	C+ = {0, 4, 8 }	D+ = {2, 6, 10 }	E+ = {4, 8, 0 }
減三和絃	C- = {0,3, 6}	D- = {2,5, 8}	E- = {4,7, 10}
大七和絃	CM7= {0,4,7,11}	DM7= {2,6,9,1}	EM7= {4,8,11,3}
小七和絃	Cm7={0,3,7,10}	Dm7={2,5,9,0}	Em7={4,7,11,2}
小大七和絃	CmM7={0,3,7,11}	DmM7={2,5,9,1}	EmM7={4,7,11,3}
減七和絃	Cdim7 ={0,3,6,9}	Ddim7 ={2,5,8,11}	Edim7 ={4,7,10,1}
半減七和絃	Cm7-5={0,3,6,10}	Dm7-5={2,5,8,0}	Em7-5={4,7,10,2}
屬七和絃	C7 ={0,4,7,10}	D7 ={2,6,9,0}	E7 ={4,8,11,2}
增七和絃	C7 ={0,4,8,11}	D7 ={2,6,10,1}	E7 ={4,8,0,3}

大三和絃	FM={5,9 ,0 }	GM={7,11 ,2 }	AM={9, 1 , 4 }
小三和絃	Fm={5, 8 ,0 }	Gm={7, 10, 2 }	Am={9 , 0 , 4 }
增三和絃	F+ = {5, 9, 1}	G+ = {7,11,3}	A+ = {9, 1, 5 }
減三和絃	F- = {5,8, 11}	G- = {7,10,1}	A- = {9, 0, 3}
大七和絃	FM7= {5,9,0,4}	GM7= {7,11,2,6}	AM7= {9,1,4,8}
小七和絃	Fm7={5,8,0,3}	Gm7={7,10,2,5}	Am7={9,0,4,7}
小大七和絃	FmM7={5,8,0,4}	GmM7={7,10,2,6}	A mM7={9,0,4,8}
減七和絃	Fdim7 ={5,8,11,2}	Gdim7 ={7,10,1,4}	Adim7 ={9,0,3,6}
半減七和絃	F m7-5={5,8,11,3}	Gm7-5={7,10,1,5}	A m7-5={9,0,3,7}
屬七和絃	F7 ={5,9,0,3}	G7 ={7,11,2,5}	A7 ={9,1,4,7}
增七和絃	F7 ={5,9,1,4}	G7 ={7,11,3,6}	A7 ={9,1,5,8}

大三和絃	BM={11,3,6 }		
小三和絃	Bm= {11,2,6 }		
增三和絃	B+ = {11,3,7}		
減三和絃	B- = {11,2,5}		
大七和絃	BM7= {11,3,6,10}		
小七和絃	Bm7={11,2,6,9}		
小大七和絃	BmM7={11,2,6,10}		
減七和絃	B6 ={11,2,5,8}		
半減七和絃	Bd7={11,2,5,9}		
屬七和絃	B7 ={11,3,6,9}		
增七和絃	B7 ={11,3,7,10}		

表 3-4 和絃對照表

以下爲和絃判斷流程圖：

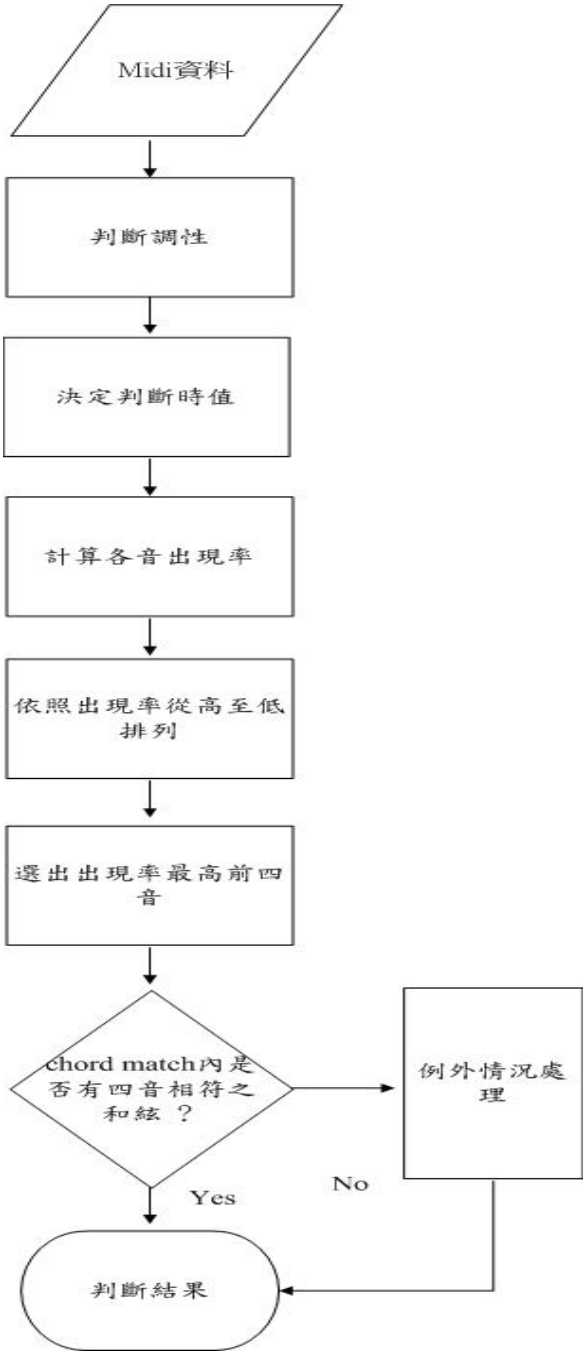


圖 3-3 和絃判斷流程圖

判斷樂曲每一個段落所屬的和絃，有幾個重要的流程 ：

1. 先辨識單拍子，複拍子，若是單拍子以兩個四分音符長度爲單位，複拍子以三個八分音符長度爲單位，將每一個判斷的單位定義爲陣列 A[ ]。
2. 依據各單位，照音高對照圖，將單位內的音，轉換成數字。
3. 計算每個音的重複出現率，將其定義爲資料結構 pitch.percent。
4. 將出現率從高至低排列，相同者則以亂數排列。

判斷方法 ：

1. 取出出現率最高的四個音。
2. 從 chord match 內尋找一模一樣的和絃。

以下列譜表爲例：

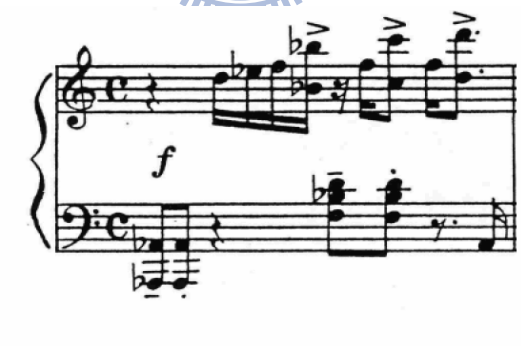


圖 3-4 Kapustin: 8 concert Etudes, op40

以陣列 A[1] 來代表此判斷單位，陣列 C[] 代表可能爲所屬和絃

Ex :

A[1].pitch={8,2,3,5,10}； // 此判斷單位中(前二拍)出現的每一個音

A[1].pitch[1].percent=4；　　// 此判斷單位中第一個音的出現率爲五

A[1].pitch[2].percent=1；

A[1].pitch[3].percent=1；

A[1].pitch[4].percent=1；

A[1].pitch[5].percent=4；

選出出現率最高前四音，分別爲 8，2，5，10，也就是五線譜上 Ab、D、F、Bb。

找出 chord match 內相對應的和聲。

A[1].match=C[Bb7] ;　　// 判斷結果爲 Bb7 和絃

四、例外情況處理　：

若無法找到與前四音相符合之和絃，則省略一音，從四音中找出任三音符合 chord match 內的三和絃。如果仍無然法找到相對應之和絃，則將第五音納入考量，若仍是和聲外音，則往後推延。若出現兩個以上可能性的和絃，則比較這些和絃，以和絃內音，同時發聲機率最高者爲優先。

下圖爲例外情況處理流程圖:

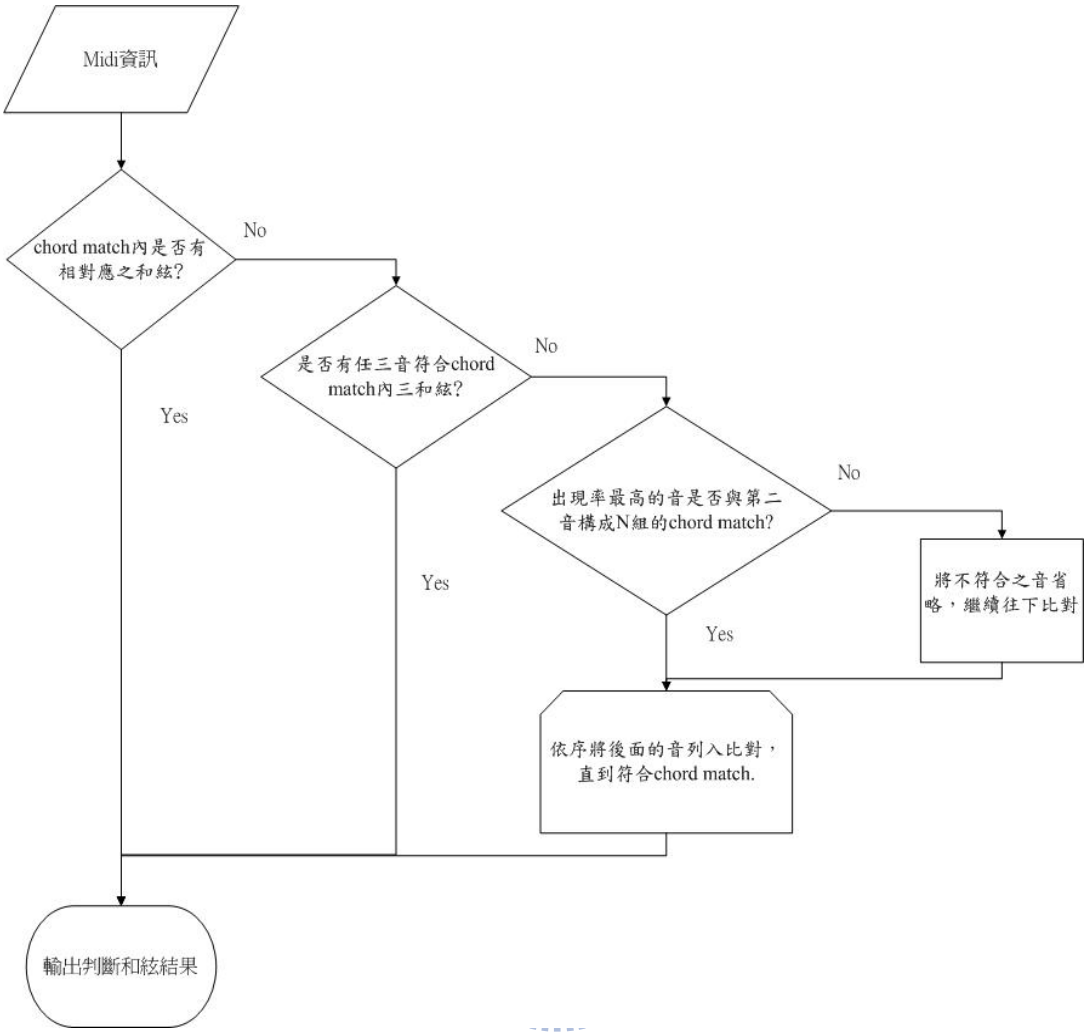


圖 3-5 例外情況處理流程圖

以下列譜表爲例：

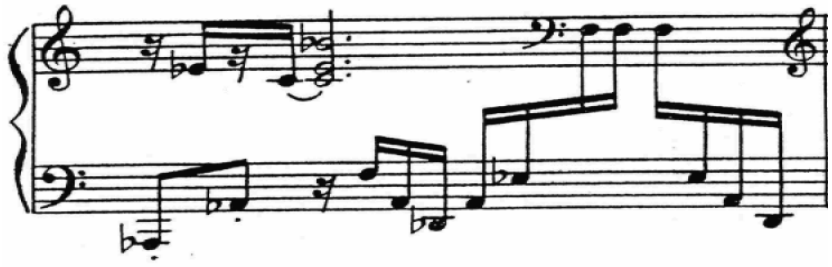


圖 3-6 Kapustin: 8 concert Etudes, op40

以陣列 A[2]來代表此判斷單位，陣列 C[]代表可能爲所屬和絃

Ex :

A[2].pitch={8,3,10,1,0,5};　// 此判斷單位中(前二拍)出現的每一個音

A[2].pitch[1].percent=3 ;

A[2].pitch[2].percent=2 ;

A[2].pitch[3].percent=1 ;

A[2].pitch[4].percent=1 ;

A[2].pitch[5].percent=1 ;

A[2].pitch[6].percent=1 ;

將出現率一樣高的使用亂數排序，選出出現率最高前四音，分別爲 8，3，10，1，無法構成任何和絃。

1. 找出 chord match 內相對應的和聲　(FALSE)
2. 選擇任三音可與 chord match 內相對應的和聲 (FALSE)
3. 從出現率最高的音開始尋找，與下一個音構成符合 N 組的 chord mapping 內之和絃

   A[2].pitch={8,3} ;
4. 往下一音，無法構成符合

   A[2].pitch={8,3, 10} ;
5. 往後推延，尋找第五音，無法構成符合

   A[2].pitch={8,3, 1} ;
6. 繼續往後推延，尋找第六音，組成 chord match 上之和絃

   A[2].pitch={8,3, 0} ;

   A[2].match=C[AbM] ;　// 判斷結果爲 AbM

五、計算各個和絃內的音類：

構成一首樂曲，和聲爲其架構，而音類，則代表著樂曲實質包含的內文，也就是旋律的部分，除了相對關係的音程距離，哪些音構成這首樂曲，也是非常重要的一部份。

音類，在音樂上意指各個八度內十二音列中的音，在音類內 C 音代表每個八度的 C，而本論文輸入的資料爲 Midi 檔案，在 Midi 編碼中，以 0 到 127 來代表所有的音高，音類則把不同八度的相同音名視爲相同，因此可以下面公式來表達。

$$PC=PN(\%12) \tag{4}$$

其中，PC 代表 Pitch Class，而 PN 則代表 Midi 編碼的 Pitch number。

然而，音高的分佈，有很大一部分決定於所屬和絃，兩者爲相對應的關係，單獨分析意義不大，因此本論文將音高的分析，以和聲爲單位，進行機率統計，下圖爲計算音類圖的流程 ：

圖 3-7 計算和絃內音類流程圖

而統計的結果，則分別將每一個和絃的音類，利用長條圖來表示，因此每一個和絃，都可以得到其相對應音類的分佈圖。

以 WC Haydn 於 1916 所作之 Ole Miss Blues 爲例，其 D7 和絃對應的音類如下圖：

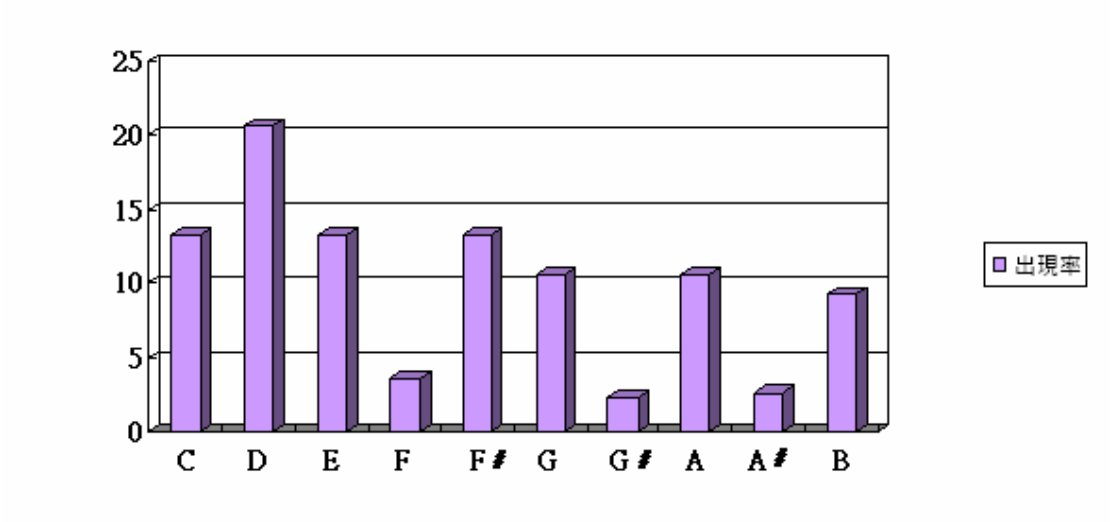


圖 3-8 音類長條圖

第二節 自動伴奏與旋律產生方法

以下爲自動伴奏與旋律產生流程圖：

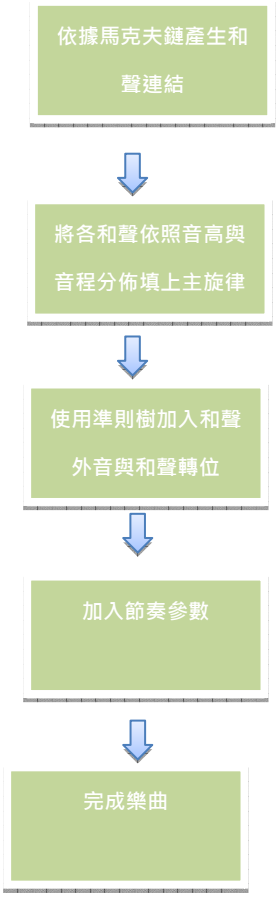


圖 3-9 自動伴奏與旋律產生流程圖

一、使用馬克夫鏈演算法產生自動伴奏:

和聲的連接，是構成一首樂曲最重要的骨幹部分，而大眾對於所謂的伴奏的概念，也是建立在和聲的連結基礎之上，本論文使用上述的分析系統採樣了大量樂曲的和聲，分析內容包括 W·C·Handy、Scott Joplin 等爵士樂大師的作品，再運用馬克夫鏈換算表，進行和聲進行機率的運算，重而產生全新的，卻類比原創樂

曲的和聲進行模式。

要說明馬克夫鏈的原理，我們可以設想在作一連串的試驗，而每次試驗所可能發生的結果定爲 $E_1, E_2, \ldots E_n, \ldots$。可能是有限也可能是無限。每一個結果 E_k，我們若能給定一個出現的可能性 p_k（即機率），則對某一特定樣本之序列 E_{j1} E_{j2} … E_{jn}，我們知道它出現的機率是 $p(E_{j1}\ E_{j2} \ldots E_{jn}) = p_{j1} \ldots P_{jn}$。這個是試驗與試驗彼此互相獨立的基本精神。

但在馬克夫鏈的理論中，我們的目的就是要擺脫「獨立」的這個假設，因此我們沒有辦法知道（也就是沒有辦法給定）每一個事件出現的可能性；也就是說「p_k」是不存在的。不過我們總得要知道一些東西才能討論，所以在馬克夫鏈中我們考慮最簡單的「相關」性。在這種情況下，我們不能給任一個事件 E_j 一個機率 p_j 但我們給一對事件 (E_j, E_k) 一個機率 p_{jk}，這個時候 p_{jk} 的解釋是一種條件機率，就是假設在某次試驗中 E_j 已經出現，而在下一次試驗中 E_k 出現的機率。除了 p_{jk} 之外，我們還須要知道第一次試驗中 E_j 出現的機率 a_j。有了這些資料後，一個樣本序列 $E_{j0}\ E_{j1} \ldots E_{jn}$（也就是說第零次試驗結果是 E_{j0}，第一次試驗是 E_{j1}……第 n 次試驗是 E_{jn}）的機率就很清楚的是 $P(E_{j0}, E_{j1}, E_{jn}) = a_j\ p_{j0j1}\ p_{j1j2} \cdots p_{jn-1jn}$。

有了以上的概念後，現在讓我們用一些數學符號來表示這些數量。在常用的術語中，用 S 來表示狀態空間，也就是一個試驗所有可能出現的狀態所成的集合。令 X_n 表示在第 n 次試驗的結果。所以 X_n 是一個隨機變數，它的取值是在狀態空間 S 中。

一組值在 S 中的隨機變數 X_0, X_1, X_2 ‥‥‥如果它滿足以下的條件，則稱爲馬克夫鏈。

$$P(X_{n+1} = x | X_0 , X_1 , X_2 , \dots , X_n) = P(X_{n+1} = x | X_n) \quad (5)$$

此式的意思是說我們以 P_{xy} 或 $P(x, y)$ 表示 $P(X_{n+1} = y | X_n = x)$，假設我們知道第 0 次到第 n 次的試驗結果，則第 n+1 次的試驗結果僅僅和第 n 次有關。

本系統先分析出和聲進行出現率，再依照馬克夫轉換矩陣，從而產生全新卻風格相似的和聲架構，以下表的和絃進行機率爲例，分析資料爲 WC Haydn 於 1916 所作之 Ole Miss Blues：

	C7	DM7	D7	D#7	E7	FM7	F7	G7-5	G7	Am7	A#M7
C7	4.26%	2.84%	1.42%	2.84%	0%	0%	2.84%	0%	1.42%	1.42%	0%
DM7	0%	1.42%	1.42%	0%	0%	1.42%	0%	2.84%	1.42%	2.84%	0%
D7	0%	1.42%	0%	1.42%	0%	0%	0%	0%	0%	0%	0%
D#7	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	5.68%
E7	0%	0%	0%	0%	0%	0%	0%	1.42%	0%	1.42%	0%
FM7	2.84%	0%	1.42%	0%	1.42%	0%	0%	0%	0%	1.42%	0%
F7	1.42%	0%	0%	0%	1.42%	4.26%	2.84%	0%	2.84%	2.84%	0%
G7-5	0%	2.84%	0%	0%	0%	0%	1.42%	0%	0%	0%	0%
G7	0%	0%	1.42%	1.42%	0%	1.42%	2.84%	0%	0%	1.42%	0%
Am7	4.26%	2.84%	0%	0%	0%	2.84%	1.42%	2.84%	0%	0%	1.42%
A#M7	4.26%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2.84%

表 3-5 和絃機率對照表

此處表格內容爲由行到列的機率。

Ole Miss Blues 之和聲連接機率矩陣:

$$\begin{bmatrix} C7_{n+1} \\ DM7_{n+1} \\ D7_{n+1} \\ D\#7_{n+1} \\ E7_{n+1} \\ FM7_{n+1} \\ F7_{n+1} \\ G7\text{-}5_{n+1} \\ G7_{n+1} \\ Am7_{n+1} \\ A\#M7_{n+1} \end{bmatrix} = \begin{bmatrix} 4.26\% & 2.84\% & 1.42\% & 2.84\% & 0\% & 0\% & 2.84\% & 0\% & 1.42\% & 1.42\% & 0\% \\ 0\% & 1.42\% & 1.42\% & 0\% & 0\% & 1.42\% & 0\% & 2.84\% & 1.42\% & 2.84\% & 0\% \\ 0\% & 1.42\% & 0\% & 1.42\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% \\ 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 5.68\% \\ 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 1.42\% & 0\% & 1.42\% & 0\% \\ 2.84\% & 0\% & 1.42\% & 0\% & 1.42\% & 0\% & 0\% & 0\% & 0\% & 1.42\% & 0\% \\ 1.42\% & 0\% & 0\% & 0\% & 1.42\% & 4.26\% & 2.84\% & 0\% & 2.84\% & 2.84\% & 0\% \\ 0\% & 2.84\% & 0\% & 0\% & 0\% & 0\% & 1.42\% & 0\% & 0\% & 0\% & 0\% \\ 0\% & 0\% & 1.42\% & 1.42\% & 0\% & 1.42\% & 2.84\% & 0\% & 0\% & 1.42\% & 0\% \\ 4.26\% & 2.84\% & 0\% & 0\% & 0\% & 2.84\% & 1.42\% & 2.84\% & 0\% & 0\% & 1.42\% \\ 4.26\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 0\% & 2.84\% \end{bmatrix}^{n} \begin{bmatrix} C7_1 \\ DM7_1 \\ D7_1 \\ D\#7_1 \\ E7_1 \\ FM7_1 \\ F7_1 \\ G7\text{-}5_1 \\ G7_1 \\ Am7_1 \\ A\#M7_1 \end{bmatrix}$$

(6)

由以上的馬克夫鏈轉換表，即可將所分析得到的和聲進行機率，重新演算而產生新的和聲進行。

二、旋律產生系統:

由以上的運算，產生自動伴奏的和聲部分以後，樂曲的和聲架構即有了基本的雛形，系統利用音類的分析採樣，將每一個和聲，對應其音類，並加入音程的控制參數，將每一個和聲填入相對應的主旋律。

以此段和聲進行爲例，對應音類之資料來源爲 W.C. Haydn 於 1916 所作之 Ole Miss Blues：

圖 3-10 和絃連接

步驟一、由各和聲所對應的音類，決定音類的比例：

D7 所對應的音類 ：

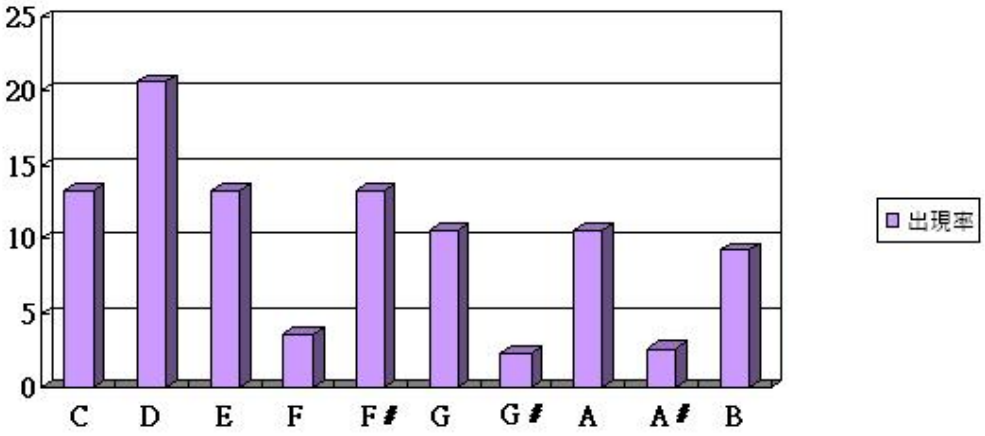


圖 3-11 "D7"對應的音類

G7 所對應的音類 ：

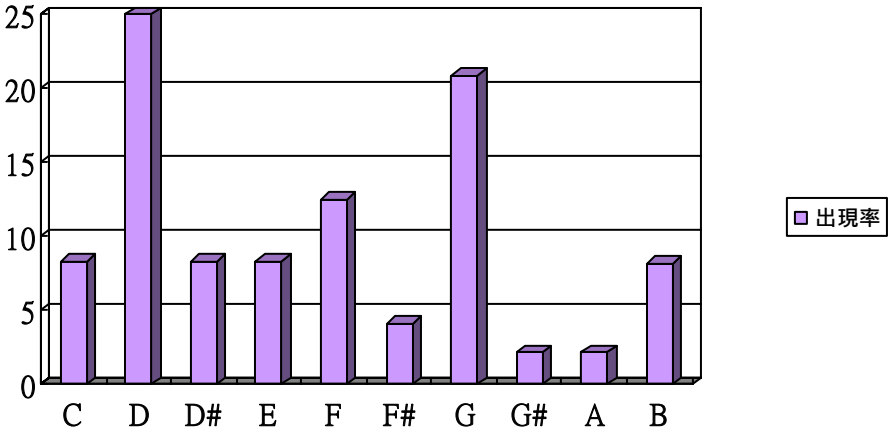


圖 3-12 "G7"對應的音類

C7 所對應的音類 ：

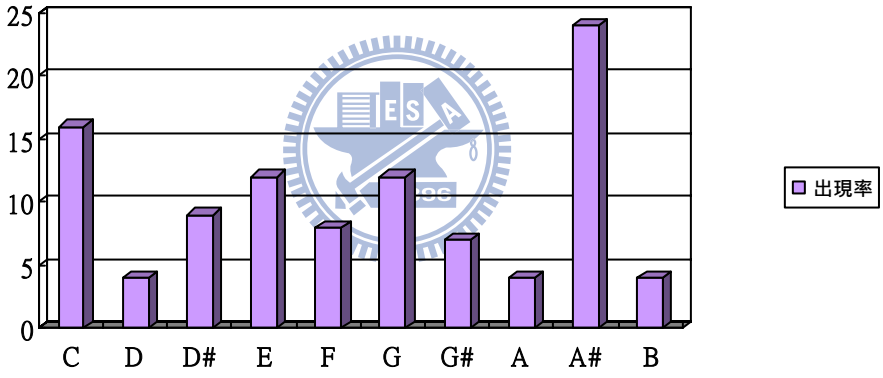


圖 3-13 "C7"對應的音類

步驟二、按照音程的出現率，將和絃內的音高重新排列：

音程	完全一度	小二度	大二度	小三度	大三度	完全四度	增四度	完全五度	小六度	大六度	小七度	大七度
%	20.07	1.941	5.161	10.9	8.836	9.29	5.409	9.496	7.267	11.56	5.863	4.17

表 3-6 音程出現率

由上述兩個步驟，即可產生和聲所對應的主旋律線條，其構成旋律的音程、和聲走向、甚至每個和聲所對應的音類，都是由分析系統採樣所得到的數據而產生，因此更可達到貼近原創精神的演算法作曲作品。

第三節 使用決策樹加入主旋律和聲外音及判斷和聲轉位

決策樹資料採掘技術，是一種以分類作爲主架構的規劃設計方式，對特定資料執行二元、三元、或多元的歸納分類，在設計時，決策樹的每一個節點 (node) 都是一個邏輯運算式，本文在判斷分析的過程中，採用決策樹中二元決策樹 (Binary Decision Tree) 來處理。

下圖爲二元決策樹：

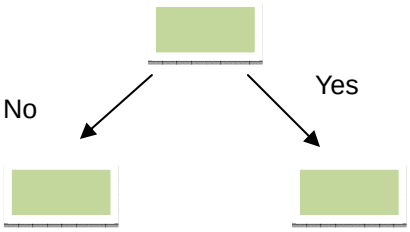


圖 3-14 二元決策樹

IF<分之節點考量運算式>

<節點第一分支運算處理>

ELSE

<節點第二分支運算處理>

END IF

在爵士樂的作曲手法中，即興部分是相當重要的，也是其風格之一大特色，爲了讓樂曲有更強烈的爵士風格，本系統利用爵士樂許多作曲的手法，使用決策樹的判斷方式，在主旋律音上，增添許多爵士型態的和聲外音，在爵士樂中，這些和聲外音構成了爵士樂曲的即興風格。

如同語言會有文法，爵士樂理上的和聲外音也有其所謂的音樂文法，本論文針對爵士樂慣用的作曲手法，將原本的主旋律音，依照下列整理出來的寫作語法，以和聲外音加以綴飾，從而產生爵士風味的旋律進行，此種方式也相當於自動作曲的音樂文法。

而聲部導進 (Voice leading) 部分，在分析系統雖然沒有做和聲轉位的判

別，但產生樂曲中，爲增加音樂的色彩及靈活度，本論文也使用決策樹，做和聲轉位的判斷，並將七和絃，額外增添了九和絃與十三和絃的色彩，從而產生豐富的和聲變化，以及低音聲部級進的線條 (Bass Line) 。

一、使用決策樹加入主旋律和聲外音 (Approach tone) :

所謂的和聲外音，泛指在音樂中，不屬於和聲內結構的音，和聲外音在增加旋律的流動性，構成旋律型，透過和聲外音不諧和的音程，解決到和聲內音 (target note) 的協和音程，增強和聲的色彩與緊張度。

以下爲使用決策樹判斷產生主旋律和聲外音即興流程圖 :

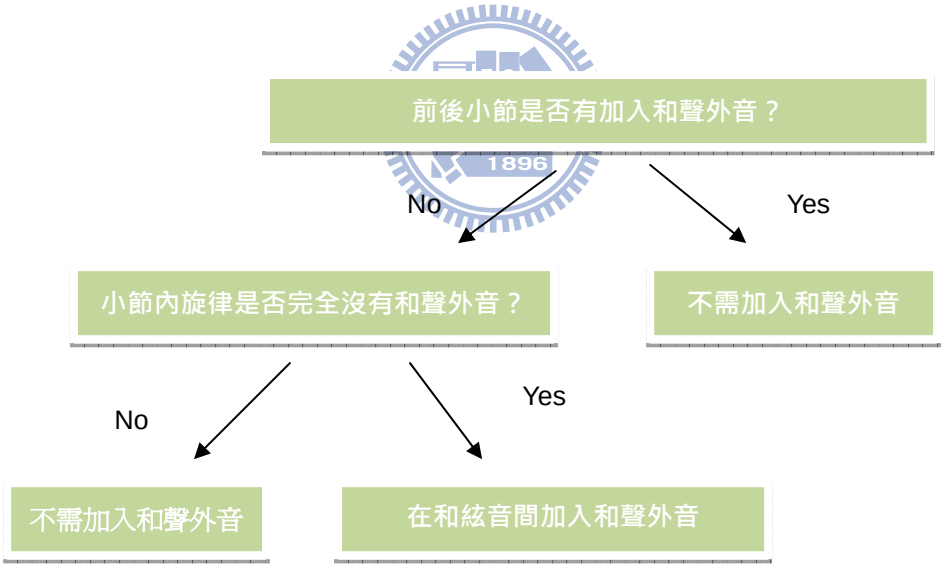


圖 3-15 旋律外音產生決策樹

以下簡介本論文使用的各種爵士樂和聲外音 [Ted & Ken, 2005] ：

1. Chromatic Approach (ch): 在和絃音前後添加半音，例如：

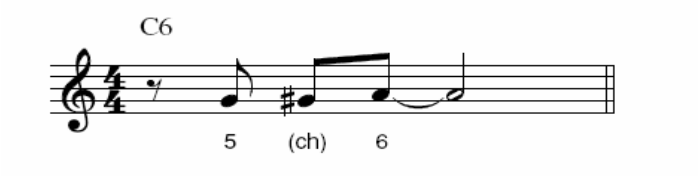


圖 3-16 Chromatic Approach

2. Scale Approach :依循爵士的藍調音階，強調「藍調音」的運用，將此大量運用在和聲外音中，例如：

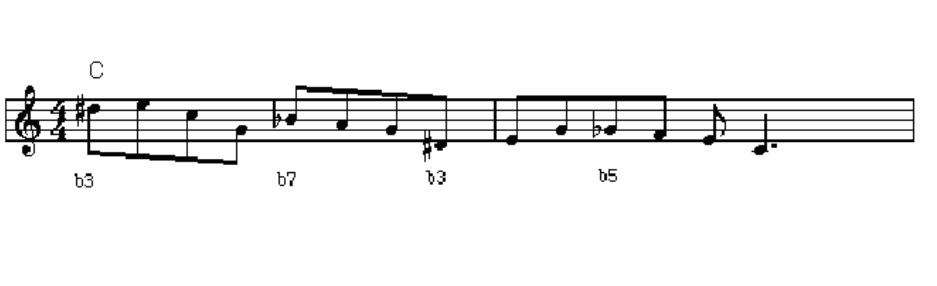


圖 3-17 Scale Approach

3. Double Chromatic Approach：在原來的半音前再加一個半音，製造 (Passing Note) 的特質，例如：

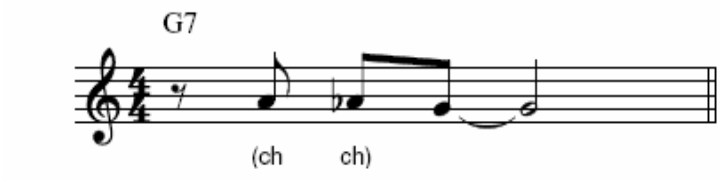


圖 3-18 Double Chromatic Approach

4. Indirect Resolution：和聲音的前面第三音加入上下階內音或半音，前面第二音加入第三音的上下半音

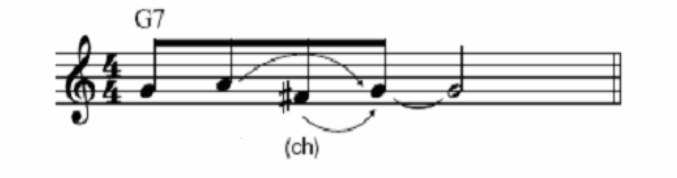


圖 3-19 Indirect Resolution

由以上的爵士樂進行法則中，配合演算法樹所使用的判斷方式，本系統可將由馬克夫鏈所演算出的和聲進行，用上述的語法加以重新詮釋，用和聲外音來創造即興效果，進而產生更貼近爵士風味的主旋律進行。

二、和聲轉位:

爵士鋼琴很重要的一部分就是聲部導進，多種和聲轉位的可能性，能決定一首樂曲的風格與質感，本論文分析得到的和聲數據，都是以原味爲主，並無考慮轉位的問題，然而真正樂曲的呈現，卻必須善用轉位和絃，才有辦法適度表現出爵士音樂的聲響。

所謂的和聲轉位包括三和絃與七和絃的轉位，在三和絃裡，每一個原位三和弦，都會擁有它自己的根音、三音和五音。根音就是最低音，而三音以及五音則是和弦音與根音之間的音程。轉位的意思就是說，以同樣的和弦音，改以另外的方式排列，可能將該三和弦的三音或五音放在最下面，這時我們就稱此新的和弦爲轉位後的三和弦。

下圖爲 CM 的轉位和絃:

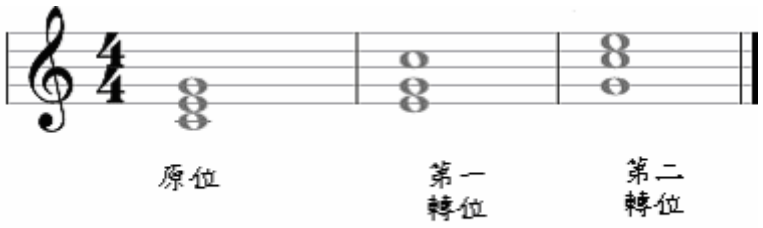


圖 3-20 轉位和絃

由於爵士樂具有很大的特色爲高數目和絃，包括九和絃與十三和絃，本系統在七和絃的轉位處理方面，爲了增加樂曲的靈活度以及爵士風味，系統在七和絃中增添九音及十三音，製造爵士樂特有的不協和感，唯需注意，七和絃若要改成九和絃，須省略根音；若改爲十三和絃，須省略五音，但仍包含九音。

以下譜例爲 C 調七和絃、九和絃、與十三和絃:

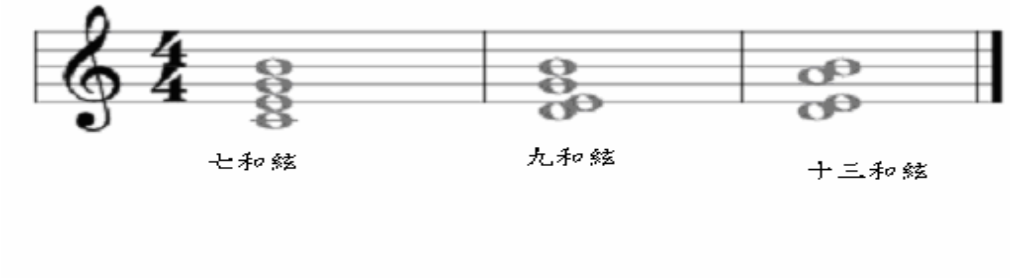


圖 3-21 高數目和絃

以下譜例爲 C 調九和絃的轉位:

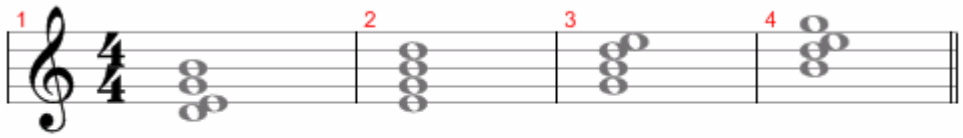


圖 3-22 九和絃轉位

以下譜例爲 C 調十三和絃的轉位:

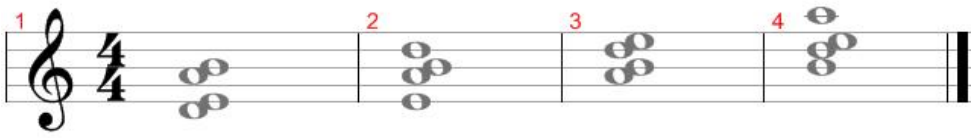


圖 3-23 十三和絃轉位

轉位和絃不但改變了樂曲的聲響與質感，並且，經由轉位和絃，也同時決定了最低聲部的音高走向，構成了爵士樂裡最相當重要的低音聲部 (Bass line) 部分，加入轉位和絃的基本方式，則以低音聲部級進爲原則。

下圖爲和聲轉位判斷方式的決策樹:

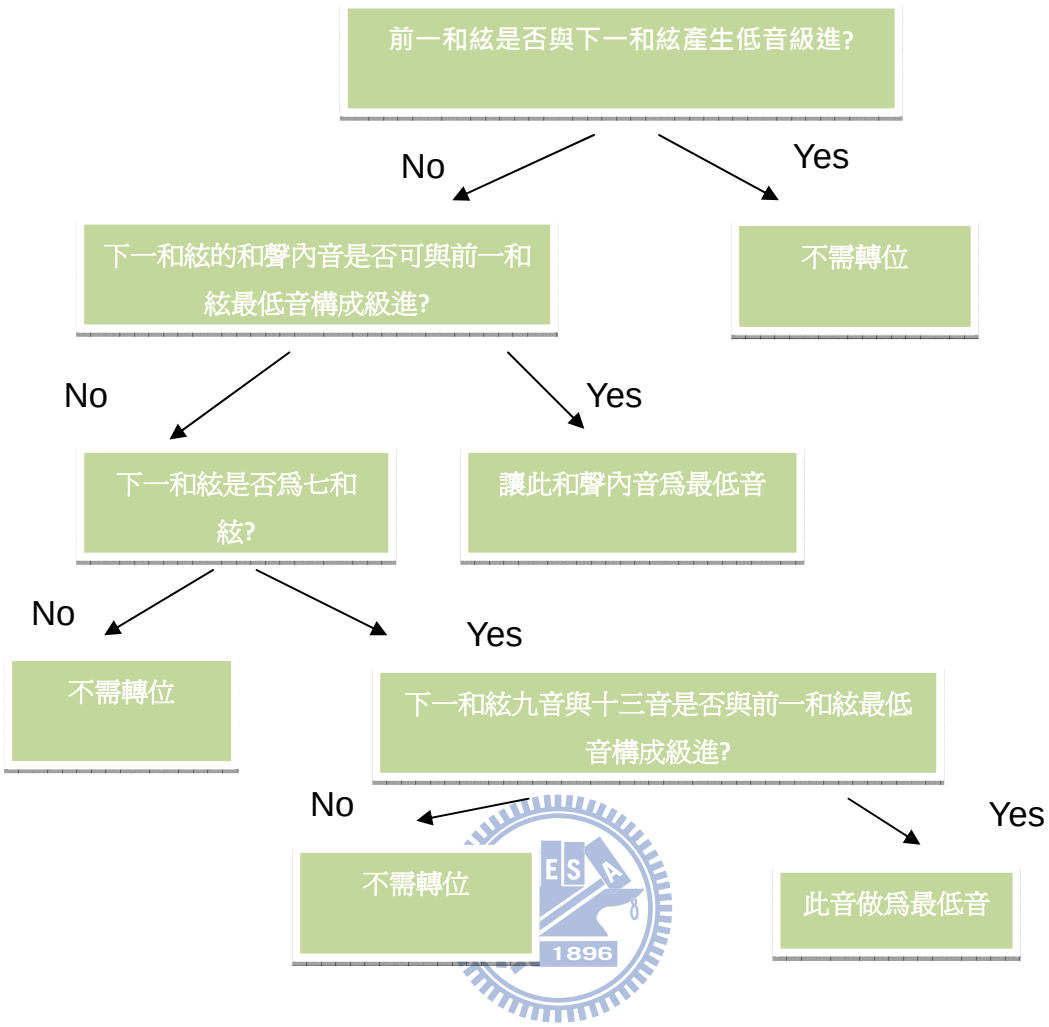


圖 3-24 和聲轉位判斷決策樹

第四節 節奏參數:

由於節奏是爵士樂的靈魂，因此爲貼近爵士風格，本論文將節奏參數以資料庫的方式建檔在程式內，以保持節奏型態的完整性。以下爲節奏參數產生流程圖:

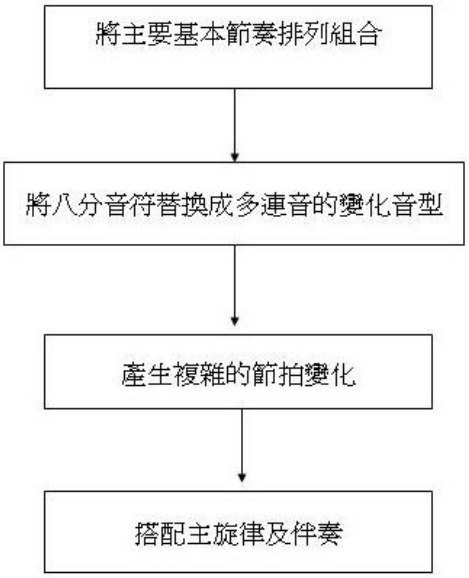


圖 3-25 節奏產生流程圖

爵士樂的節奏型態，以搖擺八分音符與切分節奏爲基礎。所謂的搖擺八分音符，指的是在過去習慣的古典樂中的記譜法，兩個八分音符等分一拍，而在爵士樂中的兩個八分音符，則是用 2：1 的比例等分一拍。

音長的時值如下圖所示:

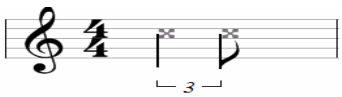


圖 3-26 八分音符時值

以下爲搖擺八分音符的基本節奏型態，所有的八分音符都以上述的時值演出：

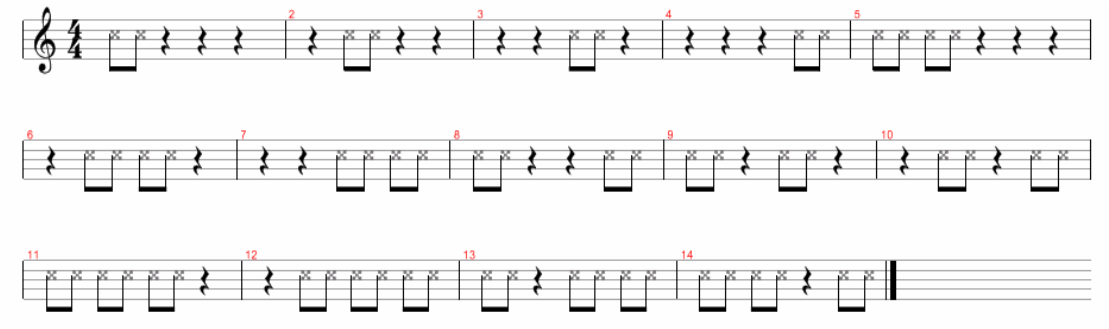


圖 3-27 基本節奏

而切分節奏有兩種變化節奏的手法：先現 (Anticipation) 與延後 (Delay) ：

1.先現：

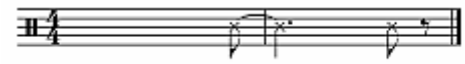


圖 3-28 Anticipation

2.延後 ：

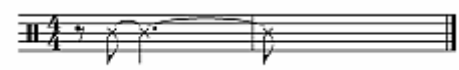
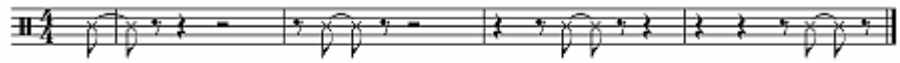


圖 3-29 Delay

由以上的基本節奏，利用這兩種手法，造成強烈的切分效果，下述的十一種變形是由以上所組合而成 [Bob, 2006] ：

(1)

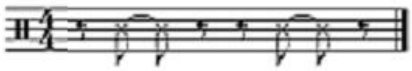


(2)

(3)

(4)

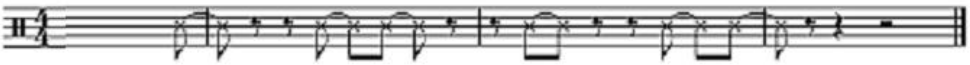
(5)



(6)

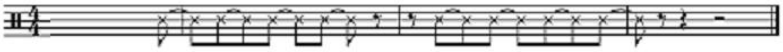
(7)

(8)



(9)

(10)



(11)

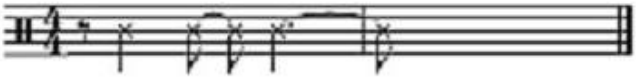


圖 3-30 變形節奏 11 種

由上述的十一種節奏型態，旋律節奏的部分，再加上基本節奏的變化，可以產生 25！的排列方式；本論文以這些 25!的排列作爲節奏的基礎骨幹，爲了增加爵士樂豐富的節奏，本系統參考原創樂曲的各種節奏變化，讓所有的八分音符二連音，又可以用六連音，三連音，以及九連音來進行替換，從而產生更豐富的變化方式，本論文將節奏參數寫入資料庫內，藉以產生拍子的變化。

第四章、研究結果與數據：

本論文由兩個系統所組合而成：其一爲樂曲分析系統，利用機率與本論文所定義的公式及運算式，分析出爵士樂曲的各個參數，包括音類、音程、以及和聲；其二爲樂曲生成系統，利用分析系統所得到的大量數據，先使用馬克夫鏈的轉換表，產生和聲的進行，也就是樂曲的架構，再利用音程與音類的數據，爲每個和聲填上主旋律的部分。

透過本系統，可得到樂曲的參數包括音程機率、音類、和聲進行與馬克夫鏈轉換表。分析資料包括 W·C·Handy、Scott Joplin 爵士樂大師的作品，下表爲系統分析所採用的作品：

作品名稱	作者	創作年份
Great Crush Collision March	Scott Joplin	西元 1897 年
Maple Leaf Rag	Scott Joplin	西元 1899 年
Cascades	Scott Joplin	西元 1904 年
Eugenia	Scott Joplin	西元 1905 年
Gladiolus Rag	Scott Joplin	西元 1907 年
Heliotrope Bouquet	Scott Joplin	西元 1907 年
Pine Apple Rag	Scott Joplin	西元 1908 年
Sugar Cane	Scott Joplin	西元 1908 年
Solace	Scott Joplin	西元 1909 年
Memphis Blues	W.C Haydn	西元 1912 年
Yellow Dog Blues	W.C Haydn	西元 1914 年
St. Louis Blues	W.C Haydn	西元 1914 年

Ole Miss Blues	W.C Haydn	西元 1916 年

表 4-1 分析作品

第一節 問卷分析調查:

一、問卷內容與結果

本問卷將受訪者分爲三種類型，第一種爲古典音樂專長者，此類型的受訪者限定在音樂系所，指接受過專業音樂訓練的人；第二種爲爵士樂手或對爵士樂有研究之學者，第三種爲非音樂專長者，此類型無接受過古典音樂或爵士音樂的訓練，爲普遍的大眾類型。

本問卷的題目分爲四項，分別爲爵士樂的整體感相似度、和聲伴奏相似度、旋律相似度、節奏相似度的評比，分數設定爲零分到一百分之間。

以下爲受訪者資料：

Age	Male/Female	Classical Musician/Jazz Musician/Non musician
19	M	N
25	M	N
25	F	C
26	F	C
60	M	J
45	M	J

表 4-2 受訪者資料

以下爲問卷評比分數

Listener	Jazz Similarity (overall)	Harmony	Melody	Rhythm
1	86	91	89	86
2	78	85	70	95
3	80	70	85	75
4	85	83	78	80
5	95	96	93	97
6	87	85	80	84

表 4-3 評比分數

二、問卷內容分析

1. 三種受訪者之平均分數：

將所有三種受訪者的個人所有評比分數取平均值，比較各類型受訪者的喜好程度差異：爵士樂專長受訪者的平均分數爲 89.6 分；古典樂專長受訪者的平均分數爲 79.5 分；無音樂專長受訪者的平均分數爲 85。

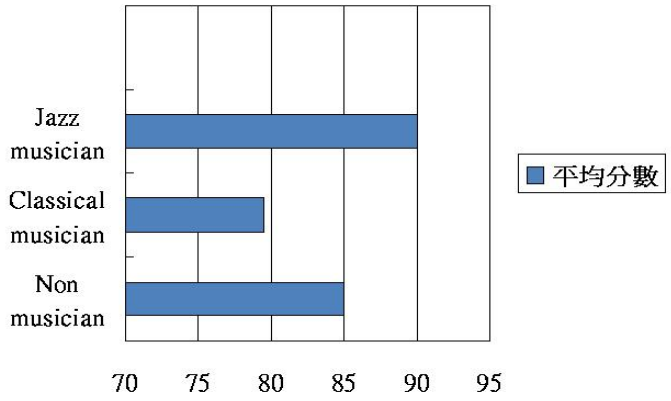


圖 4-1 受訪者平均分數

2. 非音樂專長者各項評比差異：

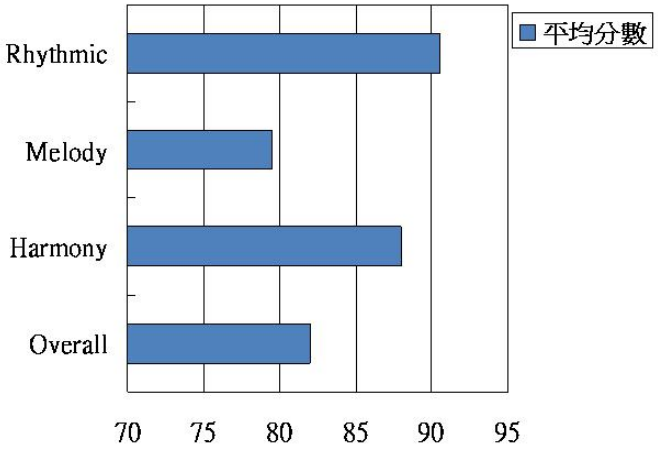


圖 4-2 非音樂專長

3. 古典音樂專長者各項評比差異：

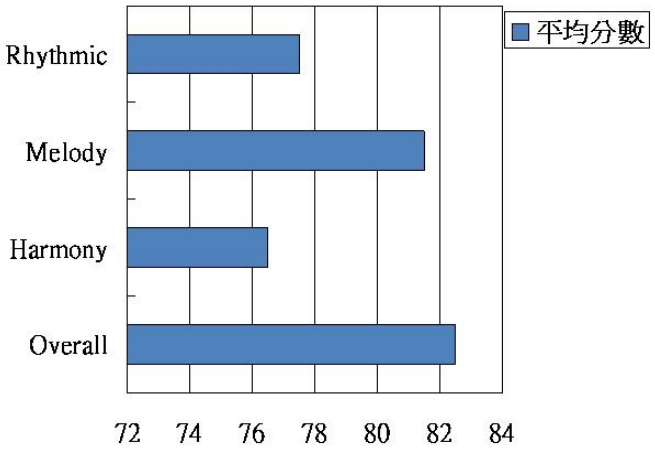


圖 4-3 古典音樂專長

4. 爵士樂專長者各項評比差異：

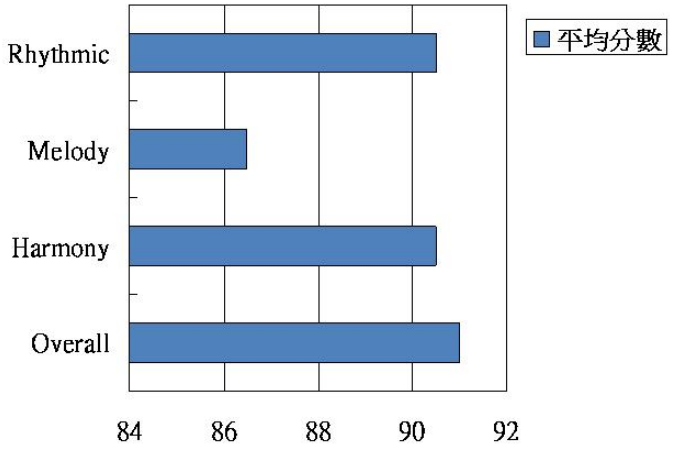


圖 4-4 爵士音樂專長

5. 各項題目評比差異：將所有受訪者的分數取平均值，比較各項所得高低。

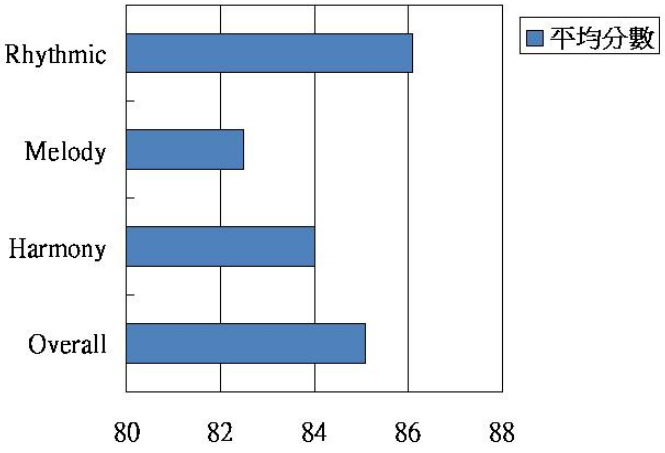


圖 4-5 各項目平均值

從上述分析，可以看出，在整體的評分中，爵士樂專長的受訪者分數最高，其次爲非音樂專長者，最後才是古典樂專長者。而在受訪者的各項評比差異中，非音樂專長給予節奏的分數最高，其次爲和聲伴奏部分；古典樂專長則給予整體感的分數最高，其次爲旋律；爵士樂專長也是給予整體感分數最高，其次爲節奏，但兩者分數差距相當小。

但以各項目平均所得分數來看，節奏部分獲得最高分數，其次爲整體感，再者爲和聲伴奏部分，最後才是主旋律。

第五章、結論：

本論文以傳統的準則作曲技術爲基礎，並額外的增添了具有分析功能的系統，提供強大的讀寫功能，並使用簡易上手的圖形化介面，藉由輸入的樂曲，去得到相似的樂曲風格寫作。

作曲是一門相當難以用數字來量化的藝術，因此，這樣一個兼具作曲與解構作曲功能的系統，能代替的，已經不再是主觀的作曲部分，而是可以經由輸入的資料，達到系統自行進步與改變的能力，讓作曲得到的內容更加客觀，也更加豐富而多元。

在實際生活的應用中，本系統可以讓許多非音樂專長的人，輕易的藉由輸入數首喜愛的樂曲，設計出具有個人風格又獨一無二的音樂作品，應用在現今的手機音樂，網站音樂，或個人相簿中，都是相當具有實用性的。在未來系統發展中，可藉由更多風格的作品輸入，而產生各式各樣的音樂型態作品；並將馬克夫鏈擴展到音高、音程、及節奏上，讓系統有更大的彈性與空間，藉由科技與藝術的結合，創造出更多元豐富的音樂。

參考文獻

Borching Su and Shyh-Kang Jeng , “Multi-Timbre Chord Classification Using Wavelet Transform And Self-Organized Map Neural Networks” , IEEE International Conference on Acoustics, Speech, and Signal Processing, (2001)

Bob, T., Keeping your Rhythmic Balance (2006)

Coker J., Improvising Jazz, 1964, reprint Fireside. (1987)

C. Ames and M. Domino. Cybernetic composer: an overview. In Understanding Music with AI: Perspectives on Music Cognition, pages 186–205, The AAAI Press/ MIT Press. (1992)

Chemillier,M., Improvising Jazz Chord Sequences by Means of Formal Grammars. (2001)

Ebcioglu, K. ,“An Expert System for Harmonization of Chorales in the Style of J.S. Bach” Ph.D. thesis, Department of Computer Science, S.U.N.Y. at Buffalo. (1986)

Johnson-Laird P., Jazz Improvisation: A Theory at the Computational Level, in: Cross I., Howell P.,West R. (eds.), Representing Musical Structure (Academic Press, 1991) 291-326

J.Oakhill (eds.), Mentals Models in Cognitive Science (Erlbaum, Mahwah, NJ, 1996)

J.A. Biles, "GenJam: Evolution of a Jazz Improviser," in Creative Evolutionary Systems, P.J. Bentley and D.W. Corne Eds. (2002) 165-187.

Joseph N. Straus, Introduction to Post-Tonal Theory (2009)

Onishi, G., Niizeki, M., Kimura, I., Yamada, H. ,A Kansei model for musical chords based on the

structure of the human auditory system, IEEE International Conference on Neural Network. (2001)

Orio, N., Déchelle, F. ,Score Following Using Spectral Analysis and Hidden Markov Models. ICMC. (2001)

Pease,T .and Pullig,K ., Modern Jazz Voicings Arranging for Small and Medium Ensembles (2005)

Rabiner, L. R., A Tutorial on Hidden Markov Models and Selected Applications. Proc IEEE, 77(2) (1989)

Steedman M.J., "A Generative Grammar for Jazz Chord Sequences", Music Perception 2 (1) (1984) 52-77.

Sheva,B., Automatic Jazz Accompaniment Computation :An Open Advice-Based Approach.(1996)

Schwarz, D., Orio, N., Schnell, N. Robust Polyphonic MIDI Score Following with Hidden Markov Models. ICMC. (2004)

Steedman M.J., The Blues and the Abstract Truth: Music and Mentals Models, in: A. Garnham, S. P. Meyn and R.L. Tweedie, Markov Chains and Stochastic Stability. (2005)

Ted, P., Ken, P. Modern Jazz voicings: Arranging for small and Medium Ensembles. (2005)

附錄

重要程式碼片段 （Auto-analysis）

```
include "playmidi.h"
using namespace std;
int node_event_array [] ={0x80,0x90,0xa0,0xb0,0xc0,0xd0,0xe0};
char *key[] = {"C","C#","D","D#","E","F","F#","G","G#","A","A#","B"};
char *chord_type[] =
{" ","m","+","dim","maj7","m7","+maj7","dim7","m7-5","7"};
char *pitch_interval[] = {"完全一","小二","大二","小三","大三","完全四
","增四","完全五","小六","大六","小七","大七"};

int chord_map[10][4] = { 0,4,7,0,    // C
                         0,3,7,0,    // Cm
                         0,4,8,0,    // C+
                         0,3,6,0,    // Cdim
                         0,4,7,11,   // Cmaj7
                         0,3,7,10,   // Cm7
                         0,4,8,11,   // C+7
                         0,3,6,9,    // Cdim7
                         0,3,6,10,   // Cm7-5
                         0,4,7,10    // C7
                         };
```

計算音程 ：

```
//=================================================================
    cv = sc_track[1];
    max_end = cv->end_time;
    stair = cv->node;
    cv = cv->next;
    while(cv){
        t = (stair - cv->node)%12;
        while(t < 0 )  t += 12;
        interval[t] ++;
        stair = cv->node;
            if(cv->end_time > max_end ) max_end = cv->end_time;
        cv = cv->next;
        total++;
```

```
    }
    for(int i = 0 ; i<12 ; i++) fout << pitch_interval[i] <<"
度:"<<setprecision(4)<< interval[i] *100 / total << "%" << endl;
    total = 0;
    fout<< endl;

      ============================================================//
```

計算各和絃內音類

```
//=================================================================
while(bar_temp){
        for(int i = 0; i < 12  ; i++) rate[i] = 0 ;
        fout <<"bar num : "<<   std::dec << bar_temp->bar_num<<" "  <<
std::dec << bar_temp->start<<" "<< std::dec << bar_temp->end<<endl;
        bar_temp = bar_temp->down;
        fout << "node : ";
        while(bar_temp){
            fout << bar_temp->node <<" ";
            music[count] = bar_temp->node % 12;
            rate[bar_temp->node%12]++;
            bar_temp = bar_temp->down;
            count ++;
        }
        //fout << endl;
        for(int i = 0; i< 12 ; i++){
            _rate = 0;
            if(rate[i] != 0 ){
                _rate = (rate[i] / count) * 100 ;
                fout <<endl<< key[i] << ":" << showpoint <<
setprecision(4)<< _rate <<"%  ";
            }

        }
==================================================================//
```

尋找是否有四個音相符合的七和絃 ：

```
//=================================================================

int check_chord(int A[],int value){
```

```
  int i = 0;
  int j = 0;
  int count = 0;
  for(i = 0 ; i < 10 ; i++){
      for(j = 0 ; j < 4 ; j++){
         if(chord_map[i][j] == A[j]) count++;
      }
      if(count == 4){
         fout <<"you find a chord: ";

         fout << key[value%12] << chord_type[i] << endl;

         return 1;
      }
      else count = 0;
  }

  return 0;

}
  ============================================================//
```

尋找是否有三個音相符合的三和絃 ：

```
//=================================================================

if(!match){
            for(int i = 0 ; i < 4 ; i++ ) buf[i] = 0; // initial buf[]
            for(int i = kind ; i > kind-4  ; i--){     // find three nodes
chore
                for(j = kind ; j > kind-4 ; j--){
                    buf[kind-j] = (nodes[j].value - nodes[i].value) %
12;
                    while(buf[kind-j] < 0)  buf[kind-j] += 12;
                }
                if(kind < 3) insertion_sort(buf,kind);
```

```
                else insertion_sort(buf,3);
                for(int r = 0 ; r < 4 ; r++){
                    zero_buf = buf[r];
                    buf[r] = 0;
                    insertion_sort(buf,3);
       // sort
                    for(int rr = 1 ; rr < 3 ; rr++ ) buf[rr] = buf[rr+1];
//rotate
                    buf[3] = 0;
                    match = check_chord(buf,nodes[i].value);
      // check
                    insertion_sort(buf,3);
      // sort
                    buf[r] = zero_buf;
                    if(match)  break;
                }
                if(match)  break;
            }
        }

        int match_point = 0;
        int root = nodes[kind].value;

//================================================================//
例外情況處理 :
//================================================================

if(!match){
            for(int r = kind ; r >= 0 ; r--){
                nodes[r].value = (nodes[r].value - root) % 12;
                while (nodes[r].value < 0 ) nodes[r].value += 12;
            }
            //for(int r = 0 ; r <= kind ; r++) cout<<nodes[r].value<<"
";
            for(int i = 9 ; i >= 0 ; i-- ){
                for(j = 0 ; j < 4 ; j++ ) {
                    buf[j] = chord_map[i][j];
                    for(int r = kind ; r >= 0 ; r--){
```

```
                        if(buf[j] == nodes[r].value){
                        match_point ++ ;
                        break;
                        }
                    }
                    if(match_point == 4){
                        fout<<"the chord you find may be : ";
                        fout << key[root%12] << chord_type[i] <<
endl<<endl;
                        harmonic[root%12][i] ++;
                        total++;
                        match = 1;
                        goto end;
                        //break;
                    }
                }
                match_point = 0;
            }
        }
        if(!match) fout << "no chord" << endl<<endl;
end:

=================================================================//
各和絃在樂曲內出現率：
//=================================================================

    for(int i = 0 ; i < length ; i++) music[i] = 0;
// zero
        for(int i = 0 ; i <= count ; i++) nodes[i].value = nodes[i].times
= 0;      // zero
        bar_temp= rv->left;
        rv = bar_temp;
        count = 0;
        match = 0;
    }
```

```
    for(int i = 0 ; i < 12 ; i++){
        for(j = 0 ; j < 10 ; j++){
            if(harmonic[i][j] != 0){
                fout<<key[i]<<chord_type[j]<<" : " << harmonic[i][j]
*100 / total << "%" << endl;
            }
        }
    }
//=============================================================
產生系統
//=============================================================
#include <sstream>

using namespace std;

class ListNode
{
    ListNode *_Prev, *_Next;

public:
    ListNode(void);

    ListNode* getPrev();
    ListNode* getNext();

    void setNext(ListNode* ptr);

    virtual int compare(const ListNode& node) = 0;
    inline bool operator < (const ListNode& node) {
        return (compare(node) < 0);
    }
    inline bool operator > (const ListNode& node) {
        return (compare(node) > 0);
    }
    inline bool operator <= (const ListNode& node) {
        return (compare(node) <= 0);
    }
    inline bool operator >= (const ListNode& node) {
```

```
        return (compare(node) >= 0);
    }
};

class List {
    bool _AutoDelete;
    ListNode *_Head, *_Current;
    int _Count;

public:
    List(bool AutoDelete = false);
    ~List(void);

    void add(ListNode* ptr);
    ListNode* remove(const int index = 0);

    int count() const;
    ListNode* moveFirst();
    ListNode* moveNext();
    ListNode* current();
    bool bol() const;
    bool eol() const;
};

class Track :
    public List,
    public ListNode
{
    int _Channel;

public:
    Track(int ch);
    ~Track(void);

    bool operator ==(int ch) const;

    int compare(const ListNode& node);
```

```
    void addNote(int tick, int pitch, int velocity, int duration);

    friend ostream& operator <<(ostream& out, Track& t);
};

class NoteEvent :
    public ListNode
{
    int _Tick, _ID, _Pitch, _Velocity;
public:
    NoteEvent(int tick, int id, int pitch, int velocity);

    int compare(const ListNode& node);
    friend ostream& operator <<(ostream& out, NoteEvent& t);
};
```

```
long swap(long d, int len) {
    union {
        long val;
        char data[4];
    } t = {d};
    if (len>4 || len<0) return 0;
    for (int i=0; i < (len >> 1); i++)
        t.data[i] ^= t.data[len-i-1] ^= t.data[i] ^= t.data[len-i-1];
    return t.val;
}
ListNode::ListNode(void) : _Prev(NULL), _Next(NULL)
{
}

void ListNode::setNext(ListNode *ptr) {
    _Next = ptr;
    if (ptr)
        ptr->_Prev = this;
}
```

```
ListNode* ListNode::getPrev() {
    return _Prev;
}

ListNode* ListNode::getNext() {
    return _Next;
}


class Midi : public List
{
    int Division, Tempo;// Division : 四分音符佔120 個Tick。Tempo : 四分音符的長度爲5000000 us (即0.5 秒)
    int Numerator, Denominator, Metronome, Notes;// Numerator - Denominator 四四拍

public:
    Midi(void);
    ~Midi(void);

    int saveToMidi(FILE* file);

    Track* findChannel(int ch);
    void addNote(int tick, int pitch, int velocity, int duration, int channel);

    static Midi& loadFromSC(stringstream & in);
};


List::List(bool AutoDelete) : _AutoDelete(AutoDelete), _Head(NULL), _Current(NULL), _Count(NULL)
{
}

List::~List(void)
{
    if (_AutoDelete)
```

```
        while (_Count>0)
            remove(0);
}

void List::add(ListNode *ptr) {//add - 把元素加進該串列中
    ListNode* p;
    if (!ptr) return;
    for (p = _Head; p && p->getNext() && *p->getNext() <= *ptr; p =
p->getNext());
    if (p == NULL)
        _Head = ptr;
    else if (*p <= *ptr) {
        ptr->setNext(p->getNext());
        p->setNext(ptr);
    } else {
        ptr->setNext(p);
        _Head = ptr;
    }
    _Current = ptr;
    _Count ++;
}

ListNode* List::remove(const int index) {//remove - 把元素自串列中移除
    int i;
    ListNode* ptr;

    if (index<0 || index>=_Count) return NULL;

    for (i=0, ptr = _Head; i < index; i ++, ptr = ptr->getNext());

    if (ptr->getPrev())
        ptr->getPrev()->setNext(ptr->getNext());
    else
        _Head = ptr->getNext();
    if (_Current == ptr)
        _Current = (ptr->getNext()) ? ptr->getNext() : ptr->getPrev();
    _Count --;
```

```
    if (_AutoDelete)
        delete ptr;

    return ptr;
}

int List::count() const {//count - 所含的元素數量
    return _Count;
}

ListNode* List::moveFirst() {
    return _Current = _Head;
}
ListNode* List::moveNext() {
    return _Current = (_Current) ? _Current->getNext() : NULL;
}
ListNode* List::current() {
    return _Current;
}
bool List::bol() const {//bol - 是否位於串列的開頭
    return (_Current == _Head);
}
bool List::eol() const {//eol - 是否位於串列結尾
    return (_Current == NULL || _Current->getNext() == NULL);
}
Midi::Midi(void) : List(true), Division(120), Tempo(500000),
Numerator(4), Denominator(4), Metronome(24), Notes(131)
{
}

Midi::~Midi(void)
{
}

int Midi::saveToMidi(FILE *file) {
    return 0;
}
void Midi::addNote(int tick, int pitch, int velocity, int duration, int
```

```
channel){
}

=============================================================//
```