國 立 交 通 大 學

電信工程學系

碩士論文

雙連語言模型預查
之大詞彙連續語音辨識系統
Bi-Gram Language-Model Look-Ahead LVCSR

研 究 生：李庚達

指導教授：陳信宏 博士

中華民國九十七年八月

雙連語言模型預查之大詞彙連續語音辨識系統

Bi-gram Language-Model Look-Ahead LVCSR

研 究 生：李庚達　　Student：Geng-Da Li

指導教授：陳信宏 博士　　Advisor：Dr. Sin-Horng Chen

國 立 交 通 大 學

電 信 工 程 學 系

碩 士 論 文

A Thesis

Submitted to Departmant of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Communication Engineering

August 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年八月

雙連語言模型預查之大詞彙連續語音辨識系統

研 究 生：李庚達　　指導教授：陳信宏 博士

國立交通大學電信工程學系碩士班

中文摘要

本系統使用的詞典具有六萬條詞彙，在動態規劃搜尋演算法上，使用雙連語言模型預查的方式，建立成連續語音辨識系統。在以 Treebank 為測試語料的實驗中，字元的辨識率可以超過 85%，若改用 TCC-300 作為測試語料，雖然沒有辦法達到同樣高的辨識率，但仍然有一定水準，故可將本系統使用於一般非特定語者的應用上。

本文中除了簡單回顧常見的一些辨識方法外，將會詳細介紹本系統所使用的辨識方法，其相關的概念，以及實作時的各項細節；除此之外，會特別著重在語言模型預查的說明上，最後再以各種實驗數據作為依據，說明本系統的各項優缺點，並分析語言模型預查的效能，並提出系統可能的改善方向。

Bi-Gram Language-Model Look-Ahead LVCSR

Student: Geng-Da Li Advisor: Dr. Sin-Horng Chen

Department of Communication Engineering
National Chiao Tung University

Abstract

In this thesis, a large vocabulary continuous speech recognition (LVCSR) system is constructed. The system is based on dynamic programming algorithm and language model look ahead algorithm; besides, a lexicon of 60,000 Chinese vocabularies and bi-gram language model are used. In the outside test of Treebank and TCC300, the character accuracy is competed with the result of HTK, and therefore the system is capable of speaker independent recognition.

In the thesis, several speech recognition algorithms are discussed, especially the dynamic programming algorithm and viterbi beam searching algorithm. Furthermore, language model look ahead algorithm is a main topic as well, and it will be interpreted thoroughly. In the end, some testing results are discussed, and a few ideas are suggested to improve the accuracy and reduce the runtime of the system.

目錄

表目錄

圖目錄

第一章、緒論

1.1 研究動機

大詞彙連續語音辨識（Large Vocabulary Continuous Speech Recognition；LVCSR）是近一、二十年來語音辨識上的重要問題之一，在基本的語音辨識技術逐漸成熟之後，人們試著將其應用在生活之中，要達到這個目標，有許多需要克服的困難；首先，一般的環境條件與實驗時特定假設下的條件相差甚遠，而且日常生活中的環境變化性很大，所以雜訊、收音、甚至音量等問題都可能會造成困擾；再者，每個人有不同的說話習慣、腔調等，即使是同一個人講同一句話，也不見得每次都一模一樣，特別是自發性語音（Spontaneous speech）的變化性更大；此外，中文的詞（Word）沒有很好的規則可以利用，若是詞典收錄的詞彙過少，辨識時便容易斷詞錯誤，或者是發生詞典並未收錄到正確答案的問題（也就是所謂 OOV；Out of Vocabulary），在在都增加了語音辨識的困難度。

當詞典的詞彙量增加時，首當其衝的便是計算量大增、記憶體使用量增加等問題，其次，詞典的複雜度（Perplexity）也會增加，辨識的困難度提高，正確率因此降低，本論文就是以增加辨識系統所能夠負荷的最大詞彙量為目標，希望能夠在相當短的時間內完成辨識，並且能夠有足夠好的辨識率。

1.2 研究成果

本系統使用六萬詞作為辨識時的詞典，在與 HTK（Hidden Markov Model Toolkit）[1]使用相同的聲學模型與語言模型的條件下，兩者可以得到相當的辨識率，而本系統的辨識時間約是 HTK 的1/2，若使用較佳的參數設定，辨識時間可以再減少，約只需要1/4。

1.3 章節概要

第二章會先簡介語音辨識的技術，一些常見的方法，以及本系統所使用的演算法，第三章才會詳細說明系統中的各項細節，除了語言模型預查（Language model look ahead）[2,15]，這部分將留到第四章介紹，第五章則是實驗與分析，在辨識率與辨識時間上以 HTK 為基準，測試系統的辨識能力，並觀察記憶體的使用情形，最後一章則是結論與展望，將提出一些系統未來改善的方向。

第二章、語音辨識概論

本章將先簡單介紹語音辨識的概念，並回顧幾個比較常見的方法，在第二節中，將介紹本系統所使用的流程，並對常見名詞作簡單的定義，統一其在本文中的用法。

2.1 文獻回顧

語音辨識（Speech recognition），以使用者的觀點來說，就是將聲音轉換成文字的技術，然而以實作的角度來說，如何使用語音中的各種資訊，對應找出最合適的文字，才是技術的關鍵，當中包括了聲學信號、頻譜、韻律、情緒、文法規則、語義等任何具有辨識性的資訊；換言之，結合各項已知或可以萃取出的資訊，設計出可以從詞典裡挑選答案的演算法，就是語音辨識最原始的想法，這類演算法的目標就是提升其結果的正確率。

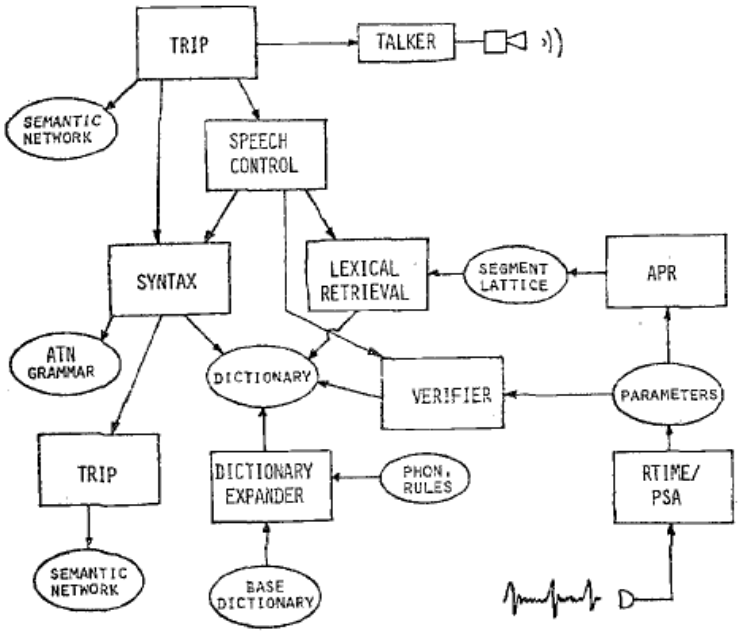

圖 2.1 HWIM 架構圖（Copyright: J.J. Wolf, and W.A. Woods [3]）

在 1980 年前後有許多相關的想法被提出，例如 HWIM（Hear What I Mean）系統（圖 2.1）[3]使用多個控制核心，分別代表文字規則、文法、句型等，將拆

解成片段的語音候選單元，拼湊成辨識的結果；Hearsay-II 系統（圖 2.2）[4]使用階層式的架構，將各種資訊彼此之間的作用方式予以規則化，透過該架構，將語音信號逐漸歸納辨識成文字；這些方法中，被認為最成功的是“統計式方法”（Statistical methods）[5]，以最大化事後機率的手段，達到語音辨識的目的，現今常用的 HMM （Hidden Markov Model；隱藏式馬可夫模型）與 N 連語言模型（N-gram language model）皆是源自於此。

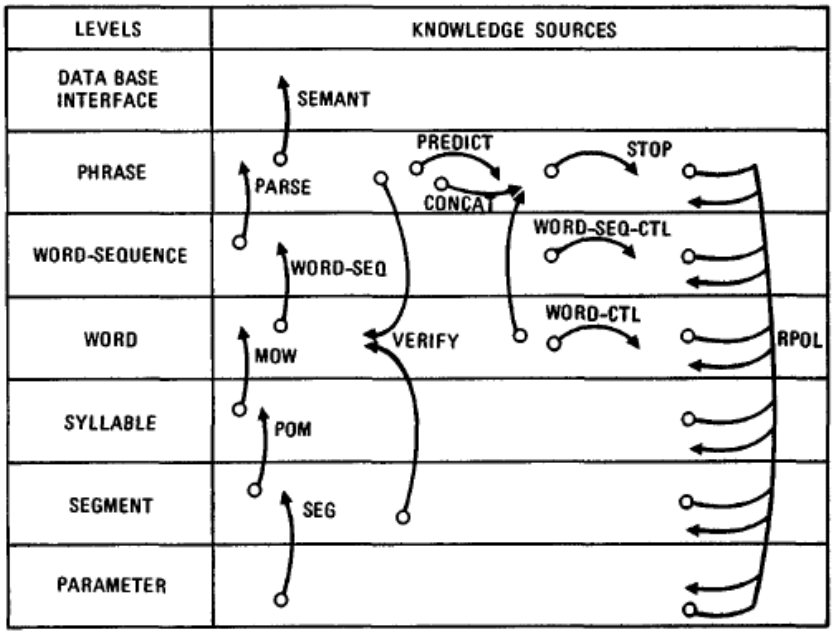

圖 2.2 Hearsay-II 系統架構圖（Copyright: L.D. Erman, et al [4]）

時至今日，語音辨識的各項技術不斷地提升，電腦的運算能力也與日俱進，原本只以少量單詞組成的句子作為辨識目標，漸漸地希望能夠辨識句法結構更複雜的長句，為了達到這個目的，詞典收錄的詞彙數量必須增加；詞典越大，選擇的複雜度就越高，運算量也隨之大增，如何提高正確率，並能夠在一定的時間內完成辨識工作，成為技術上的新挑戰。

HTK 所使用的演算法，是在 1989 年，由劍橋大學的 S. J. Young 等人所發表，名為 Token passing 的演算法[6]，在想法上應用了維特比演算法（Viterbi algorithm），在 token 上紀錄回溯資訊，最後尋找最佳路徑即為辨識結果；另外

常見的是使用有限狀態機（Finite State Machine；FSM）[7]的概念取代維特比演算法，預先將聲學模型、詞典、文法等建構在狀態機上，以語音信號作為輸入，辨識結果為輸出，建立成語音辨識系統。

本系統所使用的是 H. Ney 與 S. Ortmanns 所提出的以動態規劃法實作大詞彙連續語音辨識 [8]，圖 2.3 為其架構圖，它以動態規劃法作搜尋，利用聲學模型與語言模型當作規劃的依據，搜尋出最佳辨識結果，系統流程會先在下一節中介紹，詳細內容與辨識效果將在後面幾章詳細說明。

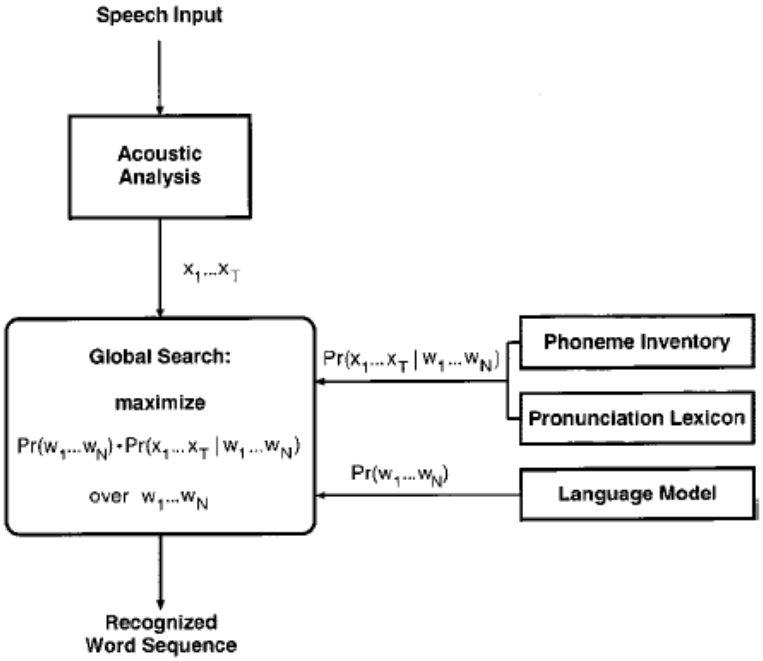

圖 2.3 以動態規劃法進行辨識（Copyright: H. Ney and S. Ortmanns [8]）

2.2 系統流程

圖 2.3 的 Acoustic analysis，會先將語音依照時間切成一個個音框（Frame），再計算每個音框的聲學特徵參數，這個步驟把語音轉換成一連串的向量參數，將語音辨識轉換成電腦可以計算的數學問題——在已知條件中，搜尋能使該串向量參數有最大相似度的連續詞串，這裡的相似度並不直接等同於 Likelihood，它包

含了聲學模型層次（Acoustic level）以及語言模型層次（Language model level）的分數，兩者會乘以適當的權重值後相加；搜尋的過程，則會使用詞典的規則等加以限制。

把語音切成音框，是相當聰明的想法，透過如此時間對位（Time alignment）的作法，不但使 HMM 和維特比演算法能夠跟語音辨識巧妙地結合，而且也一併解決了說話速度不同的問題。

根據動態規劃法演算法，每個時間點會產生新的 hypothesis，因為中文裡並無適當的翻譯，故本文中將直接以 hypothesis 稱之；位在不同 HMM 狀態（State）上的 hypothesis，會得到不同的分數，這個分數會累積到它所產生的新 hypothesis 上；hypothesis 在音節中會根據 HMM 的規則作轉移，在音節之間則是根據詞典及語言模型的規則作轉移，到最後累積分數最高者即為所求，可以靠著額外紀錄的回溯標記，找出最佳存活路徑（Survivor path），得到辨識結果。這樣的作法屬於 One-pass，可以在一次的搜尋過程中得到辨識結果，相較於此，Two-pass [13,14] 的作法會先產生詞圖（Word graph），然後再藉由語言模型等手段從詞圖中找出辨識結果，其優點是可以將更複雜的語言模型應用在辨識上。

詞典的存在相當於建構了辨識時的搜尋空間（Search space），它的資料結構可以隨著需要而有所改變，若直接將所有的詞彙一一完整地建立，便是最簡單的是線性詞典（Linear lexicon）；把線性詞典中，各詞彙相同的前綴（Prefix）合併，可以得到樹狀的詞典結構（Tree-structured lexicon），稱之為詞典樹（Lexicon tree），這也是本系統所採用的結構；在 HTK 中，則將詞彙與語言模型合併建立成一個詞網（Word network）；若是以有限狀態機（Finite state Machine）為基礎的辨識系統，則會在建立狀態機時，一併加入詞彙的資訊。上述的是幾種常見的作法，不過原則上詞典的資料結構並無所謂好壞之分，純粹視系統的需要而有所差異。

第三章、演算法介紹

本系統在辨識中使用了聲學模型、詞典、語言模型的資訊，在本章中將依序介紹；三者在運作時雖然互相獨立，但是產生的資訊會被巧妙地安排加入到演算法中，最終完成辨識的目的，提高了只用單獨一項資訊的辨識率；此外，維特比演算法與其它數種常用的刪除演算法（Pruning algorithm）在系統中擔任的角色為何，也將會一併介紹。

3.1 聲學模型層次

先對輸入語音以音框為最小單位求取參數，包括 12 維的梅爾倒頻譜參數（Mel-Frequency Cepstral Coefficients；MFCCs）、12 維的 delta-MFCCs、1 維的 delta-Energy 參數、12 維的 delta-delta-MFCCs 以及 1 維的 delta-delta-Energy 參數，共計 38 維；然後計算該參數與 HMM 中的各個狀態的相似度（Likelihood），

$$\begin{aligned} b(x_1, x_2, ..., x_N) &= \sum_k c_k N(\mathbf{x}; \mu_k, \Sigma_k) \\ &= \sum_k c_k \cdot \frac{1}{\sqrt{(2\pi)^N |\Sigma_k|}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k) \right) \end{aligned} \tag{3-1}$$

其中 $\mathbf{x} = (x_1, x_2, ..., x_N)^T$ 為該音框的參數向量，c_k 為 mixture weight，μ_k 及 Σ_k 分別代表 HMM 狀態的第 k 個 mixture 的平均值向量（Mean vector）以及共變異矩陣（Covariance matrix），$N(=38)$ 代表參數的維度，b 為相似度；在實作中，為了提高精確度，會先取 b 的對數值再加到 hypothesis 上，這就是所謂的聲學模型分數，如果直接拿機率值相乘的話，數值很快便會超出浮點數所能表示的範圍（在此情形下，會造成 underflow）。

3.2 詞典層次

3.2.1 右相關聲韻母模型

中文每個字的發音方式皆可對應到一或多種音節（Syllable），歸納的結果，一般將音節分為 411 類；每一個音節可以再分為聲母和韻母，辨認時如使用右相關聲母模型（Right Context Dependent Initial Model）及前後文獨立韻母模型（Context Independent Final Model），則總計有 100 類的聲母跟 40 類的韻母，我們簡稱它們為右相關聲韻母模型（RCDIF Model）。

使用 HMM 模擬右相關聲韻母模型時，分別用 3 個和 5 個狀態去描述聲母和韻母，如此每個音節便由 8 個狀態所組成。此外限制每一個狀態只能夠前進到下一個狀態或者是回到自己，不能夠任意地跳躍，如圖 3.1 所示。

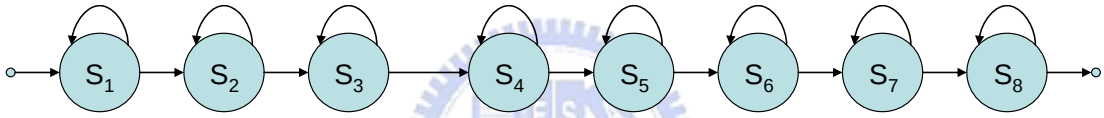

圖 3.1 音節的 HMM 示意圖

3.2.2 固定轉移機率

當執行演算法時，固定 HMM 的轉移機率（Transition probability），只考慮觀察機率（Observation probability），也就是 3.1 節所述之相似度；這樣的假設是基於以下的考量，假設某兩筆轉移機率大小分別為 0.9 與 0.1，其對數值僅差距 2.20（若以 Euler number e 為底），因此在多數的情況下，轉移機率彼此間的差距將遠小於不同 HMM 狀態間的相似度差距；既然轉移機率的影響很小，為了減輕運算的複雜度，直接假設所有可行路徑的轉移機率都一樣大。

因為固定了轉移機率的大小，所以每一次的轉移所得到的轉移機率分數皆相等，故不用再額外考慮轉移機率，因此每個音框的聲學模型分數，等於其觀察機率的對數值，也就是 log likelihood。當進入下個時間點時，各個 hypothesis 上的分數將會累加到其對應產生的新 hypothesis 上。

3.2.3 維特比演算法

根據前述的轉移規則，由t時間的hypothesis所產生的新hypothesis，可能會有重複的情形，例如圖 3.2 中t+1 時間的狀態S_{i+1}，可以是由t時間的S_i轉移產生，也可以從t時間的S_{i+1}產生。

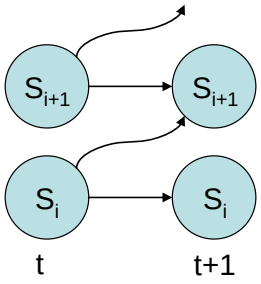

圖 3.2 音節內的 HMM 狀態轉移

這兩個不同來源的S_{i+1}，其轉移機率相等，t+1 時間的觀察機率相同，t+1 時間以後所有可能產生的路徑亦完全相同，只有在t時間以前有所差異，故只需要保留其中最佳的一個hypothesis即可；而每個hypothesis在t時間以前運算的結果已經被累加成為單一筆分數，因此當有重複的狀態產生時，直接比較其來源，保留分數較高者即可，此過程與維特比演算法的精神相同；在 3.3.3 節中，將進一步用Trellis來解釋它。

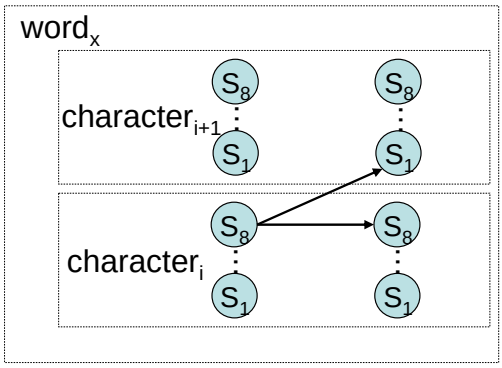

圖 3.3 字元間的 HMM 狀態轉移

每一個hypothesis可以產生若干個新的hypothesis，數量多寡將隨著以下幾種情況而不同，在圖 3.2 中的S_i可以產生兩個新的hypothesis，如果是某一個字元的第 8 個狀態，如圖 3.3 中，$character_i$的第 8 個狀態除了會回到自己以外，還可以

產生到它下一個字元的第 1 個狀態的hypothesis。

除此之外，倘若第 8 個狀態所在的字元是一個詞的詞尾，那麼它除了能回到自己外，尚可開啟一個新的詞，也就是進入一個新的詞的詞首的第 1 個狀態，其中包含了自己所在的詞，其詞首的第 1 個狀態，從圖 3.4a 可以觀察到，這個步驟可能會產生數百個到數千個新的 hypothesis，會隨著使用詞典的不同而有所差異；本系統所使用的詞典，即使已經合併成詞典樹，仍然有四千多種不同的詞首；而且系統在進行詞轉移（Word transition）時，還需要額外加入語言模型的機率（Language model probability，這部分將在 3.3 節詳細介紹），因此詞轉移時的運算量會大幅增加。

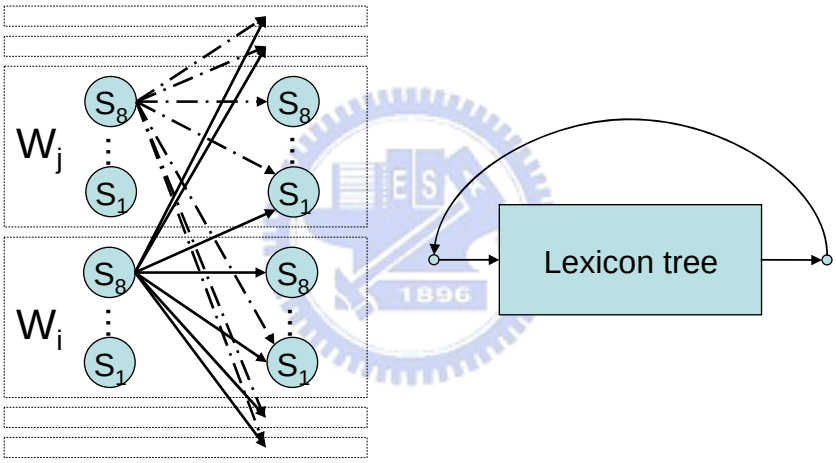

圖 3.4a 詞轉移

圖 3.4b 使用詞典樹作語音辨識

若以維特比演算法的角度來考慮詞轉移的步驟，可以想像在各詞尾的最後一個狀態額外加上一條到空狀態（Null state）的路徑，空狀態是一個假想的狀態，它代表所有詞首的第 1 個狀態，當詞尾的最後一個狀態做轉移時，它將重新進入詞典樹，也就是像圖 3.4b 所表示的概念 [9]。

3.2.4 刪除演算法

重複上面的步驟幾次以後，hypothesis 的數量將變得很龐大，特別是考慮 N 連語言模型時，增加的速度會更快（3.3.3 節會詳細解釋其原因），因此在實作上

會將分數較低的 hypothesis 刪除（Prune），其原理是因為辨識途中分數落後太多的 hypothesis，能夠再將分數重新超前，成為最佳辨識結果的機會是微乎其微；因此，每個時間只需要保留足夠數量的 hypothesis，即可保證能與不刪除的結果十分近似甚至完全相同（第五章中將以實驗驗證），而且能夠大幅地縮減辨識所需要的時間；此外，如果不作刪除的動作，hypothesis 的數量將成長到硬體難以承受，因此 pruning 對本系統所使用的演算法而言，是必要的一個步驟。

本系統考慮以下三種刪除方法：聲學刪除法（Acoustic pruning）、統計式刪除法（Histogram pruning）以及詞轉移刪除法（Word end pruning 或稱為 Language model pruning）[8]，說明如下。

執行聲學刪除法時，先找出最高分的 hypothesis，再根據事先設定的門檻值，將與最高分差距超過門檻值的 hypothesis 刪除掉，也就是一般所謂的光束搜尋演算法（Beam searching algorithm；等同於 HTK 中 HVite 的-t 參數）；統計式刪除法與前者類似，它保留了每個時間點中分數最高的 N 個 hypothesis，N 值的大小也是依據事前的設定；前面兩種方法都是在準備進入下一個音框前執行，也可以選擇只執行其中一種方法，由於聲學刪除法會明確地定義所保留的 hypothesis 的分數門檻值，因此後面章節的測試結果，除非有特別聲明，否則都是只有使用聲學刪除法。

第三種刪除方法，詞轉移刪除法，只有在詞轉移時才作用，3.2.3 節提到，當詞轉移時，一個 hypothesis 可能會產生數千個新的 hypothesis，但是倘若參考語言模型所提供的資訊，便可以直接篩選出當中較佳的 hypothesis，直接刪除掉較差的 hypothesis（等同於 HTK 中 HVite 的-v 參數）；至於語言模型如何提供資訊，將在 3.3 節中詳細說明。

3.3 語言模型層次

3.3.1 N 連語言模型

本系統使用較基本的 N 連語言模型，一般以下式表示：

$$P\left(W_i \mid W_1^{i-1}\right) \approx P\left(W_i \mid W_{i-N+1}^{i-1}\right) \tag{3-2}$$

W_1^{i-1} 表示 $W_1\, W_2 \ldots W_{i-1}$ 共計 $i-1$ 個詞的有序詞串，左式表示在 W_1^{i-1} 詞串後出現 W_i 詞的條件機率，右式則為左式根據 $N-1$ 階的馬可夫假設化簡後的結果，從式中可以知道它只與其前 $N-1$ 項相關。以 3 連（3-gram；Tri-gram）語言模型為例，n 個詞的詞串機率可以下式表示：

$$\begin{aligned} P\left(W_1^n\right) &= P\left(W_1\right) P\left(W_2 \mid W_1\right) P\left(W_3 \mid W_1^2\right) P\left(W_4 \mid W_2^3\right) \cdots\cdots P\left(W_i \mid W_{i-2}^{i-1}\right) \\ &= P\left(W_1\right) P\left(W_2 \mid W_1\right) \prod_{i=3}^{n} P\left(W_i \mid W_{i-2}^{i-1}\right) \end{aligned} \tag{3-3}$$

$P\left(W_1\right)$ 因為沒有前詞（Predecessor），故以單連（1-gram；Uni-gram）語言模型代替，$P\left(W_2 \mid W_1\right)$ 只有一個前詞，故以雙連（2-gram；Bi-gram）語言模型代替，其餘末項皆可由 3 連語言模型求得，$P\left(W_1^n\right)$ 就是該詞串 W_1^n 的語言模型分數總和。

想像某個 hypothesis，其存活路徑（Survivor path）在 W_1^i 上，則該 hypothesis 上的分數，除了來自聲學模型的分數以外，還有上述的語言模型的分數；在測試時，會先將語言模型的分數乘上一個權重值後，再加到 hypothesis 中，不同的權重值會得到不同效果的辨識率，從這一點可以看出語言模型對本系統所使用的演算法的影響性（事實上，語言模型的權重值對 HTK 的辨識結果也有相當的影響力，在第五章的實驗中可以明顯地觀察到這個現象；可以參考 HVite 的-s 參數）；更多的測試結果，將會在第五章中討論。

3.3.2 語言模型預查

由於使用詞典樹的緣故，必須在 hypothesis 作詞轉移時才能知道前面那一段路徑是走在哪一個詞裡，這時才能得到 W_i，然後從語言模型裡求得

$P(W_i \mid W_{i-N+1}^{i-1})$；以圖 3.5 中的三個詞，“交通”、“交通部”及“交通大學”，為例（詞尾以雙層圓圈表示），當 state 進入詞首“交”時，尚無法加入語言模型的分數，即使進入了“通”，仍舊無法判斷該加上何者的分數，必須等到 hypothesis 要從“通”的最後一個狀態離開時，方能在轉移出去的 hypothesis 上加上W_i＝“交通”的語言模型分數，至於“交通部”與“交通大學”兩詞的語言模型分數，最快也要在進入“部”或“大”時才能加入（因為在它們下面沒有其他分支，故能提早）。

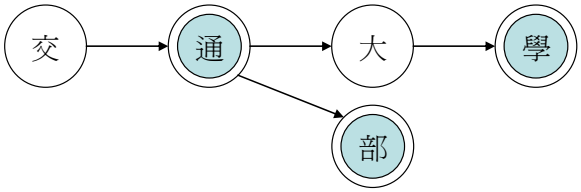

圖 3.5 詞典樹簡圖

如果能夠在剛進入一個分支時，就提供適當的語言模型分數的話，便能保護語言模型中出現機會高的詞串，提高它存活的機率，可以進一步地縮小 beam width 的寬度，減少辨識所需要的時間而又不會影響到辨識率；在上一段的例子中，可以在進入“交”時產生 3 個 hypothesis，分別代表三個詞，這樣的作法相當於線性詞典，缺點是，詞轉移的步驟會產生與詞彙數目等量的 hypothesis（對本系統而言，由四千增加為六萬），反而大幅地增加 hypothesis 的數量，增加辨識的時間；若使用語言模型預查的演算法，則不需要增加額外的 hypothesis 數量，並且能夠在進入各詞首或詞典樹上的分歧點時，提供可靠的語言模型資訊，3.2.4 節提到的詞轉移刪除法機制，便是以此做為篩選 hypothesis 的依據，因此若是使用樹狀詞典卻不作語言模型預查，就不能夠使用詞轉移刪除法；但是使用語言模型預查也會增加額外的運算量及記憶體使用量，關於這兩點以及它的詳細演算法，將在第四章有更進一步的分析。

3.3.3 維特比演算法

前面提到，當 HMM 狀態轉移時 hypothesis 會互相競爭，只有最佳來源者能存活，換句話說，任何一個時間點中，任何一個詞的任意字元中的任何一個狀態，最多只會有 1 個代表它的 hypothesis 存活，這也是維特比演算法的特性之一；然而，在考慮語言模型之後，相同詞彙中同一字元同一狀態，但具有相異前詞（Predecessor）的 state，將不會互相競爭，而是都會存活，不過，相同詞彙、相同字元、相同狀態且具有相同前詞的 hypothesis，仍將彼此競爭，最多只能存活一個；也就是說，原本被視為相同的 hypothesis，如果彼此具有相異的前詞，則會被視為不同，因此就不會互相競爭；倘若仍具有相同的前詞，那麼還是會互相競爭，只留下最佳的一個。

以 Trellis 的觀點來看，原本 Trellis 中的節點（Node）數量等於下式：

$$Node_{Trellis,noLM} = Node_{LexiconTree} \times State_{Syllable} \tag{3-4}$$

其中，$Node_{Trellis,noLM}$ 為不考慮語言模型下 Trellis 的節點數，$Node_{LexiconTree}$ 為詞典樹的節點數，$State_{Syllable}$ 表示每個音節的 HMM 狀態數。在考慮語言模型之後，Trellis 的節點數量增為，

$$Node_{Trellis,LM} = Node_{LexiconTree} \times State_{Syllable} \times Num_{predecessor} \tag{3-5}$$

其中，$Num_{predecessor}$ 為前詞的總數量；以本系統為例，使用雙連語言模型後，$Node_{Trellis,LM}$ 將增加為原本 $Node_{Trellis,noLM}$ 的六萬倍，如果不作任何的 pruning，每個時間可能存在的 hypothesis 最大數量也增加為原本的六萬倍，這也是 3.2.4 節提到，考慮 N 連語言模型後，hypothesis 增加速度會更快的原因。

3.3.4 Super HMM

在本系統中，每個音節由 8 個 HMM State 所組成，而詞典樹是由音節所串接而成，因此可以把詞典樹想像成是以 HMM State 為節點，所構成的一棵樹，圖 3.6 為其示意圖，在樹中定義了每一個狀態的轉移規則，其節點的數量正是等

於 $Node_{Trellis,noLM}$ 。

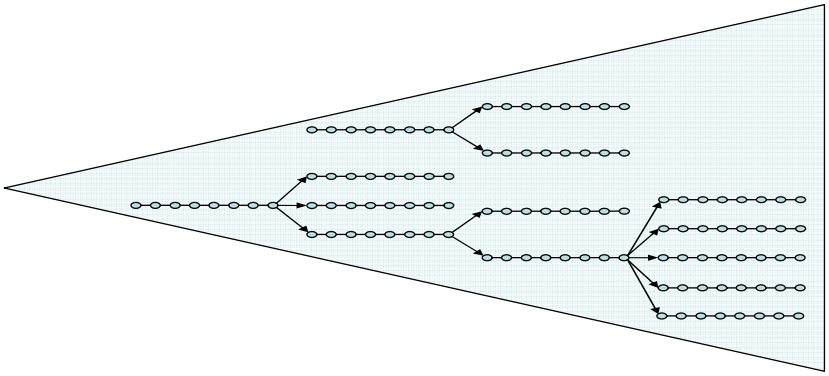

圖 3.6 由 HMM 構成詞典樹

但在加入語言模型後，這樣一棵樹只能用來表示具有某個特定前詞的詞典樹，若將其視為一棵子樹（Sub-tree），而將所有具有不同前詞的子樹組成一棵更大的樹，構成一個 Super HMM [8]，它便能定義所有各種不同前詞的狀態的轉移規則，其節點數量也增加為 $Node_{Trellis,LM}$；圖 3.7a 是一棵使用雙連語言模型的 Super HMM 示意圖（假設只有 A、B、C 三個詞），圖 3.7b 則是三連語言模型的 Super HMM 示意圖。

需要補充說明的是，本文中的“前詞”（Predecessor）一詞，並不特別單指前面一個詞彙，而是依據語言模型的需要有所不同；例如使用雙連語言模型的時候，因為語言模型已經被降到一階，所以只需要往前看一個詞就可以；若是三連語言模型的話，前詞代表前兩個詞所構成的有序詞串，因此圖 3.7b 中的總詞數雖然只有 3 個，但是卻組成不同的 9 種前詞，其所構成的 Super HMM 也將更為龐大。

事實上，在大詞彙的語音辨認系統當中，不太可能真實存在一份完整的 Super HMM，舉例來說，在使用六萬個詞彙，雙連語言模型的條件下，Super HMM 中至少有 288 億個節點，估計需要超過 100GB 的記憶體。若是使用三連語言模型，至少再大六萬倍。

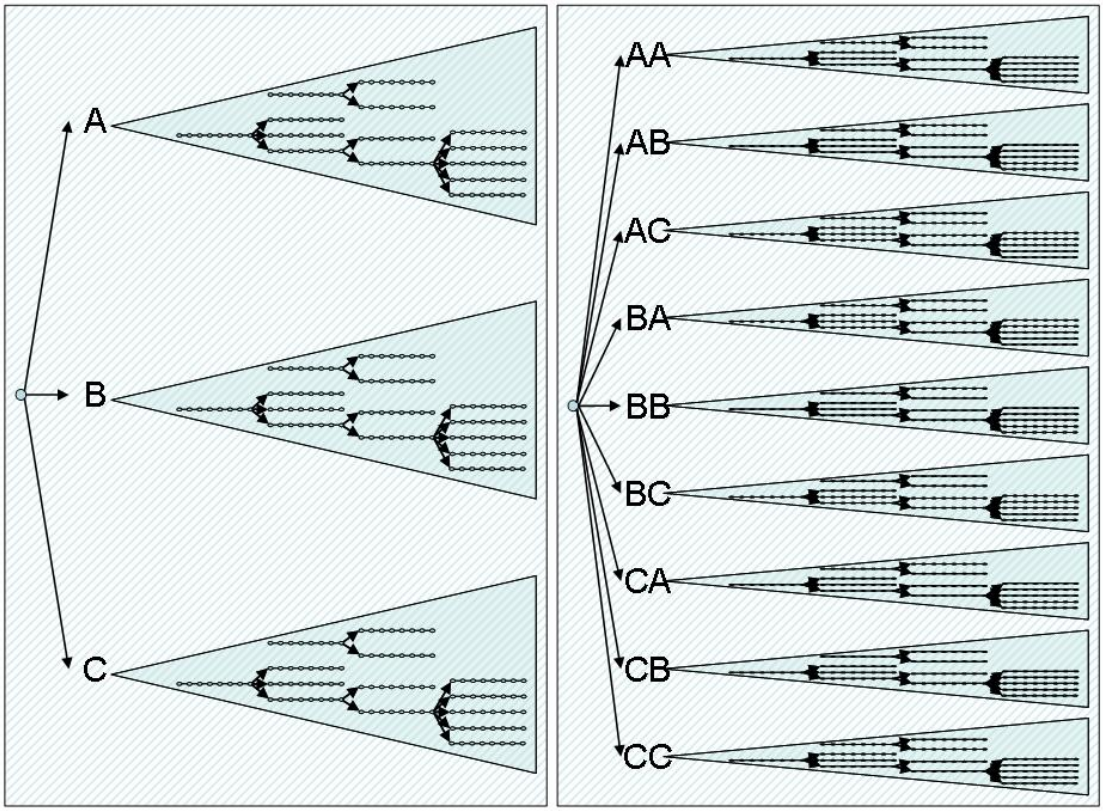

圖 3.7：(a)雙連語言模型；(b)三連語言模型的 Super HMM

3.3.5 拷貝樹的替代方案

因為不太可能真實存在一份完整的 Super HMM，因此一般的想法是使用拷貝樹（Tree Copy）的方式，動態建立需要用到的子樹，並移除不需要的子樹；本系統也是採取動態管理的概念，但在資料結構的安排上有所差異，作法不同於拷貝樹，說明如下。

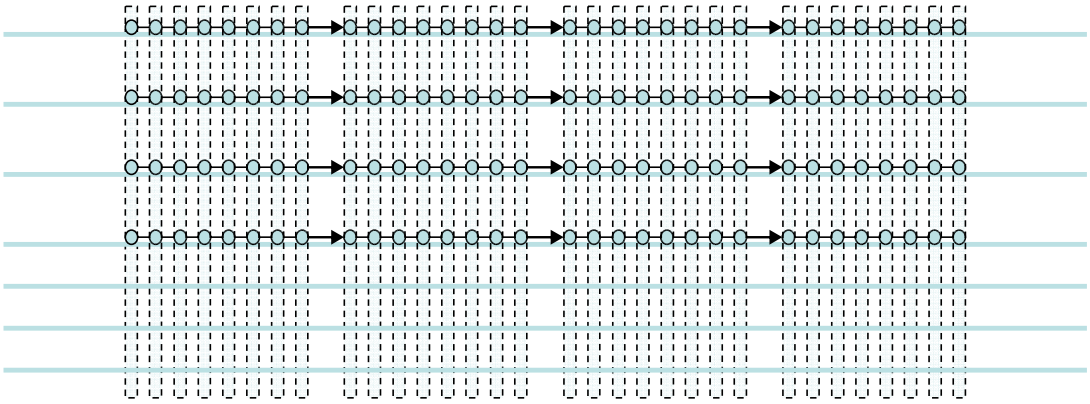

圖 3.8 將子樹重疊之示意圖

因為每一棵子樹的結構都一樣，所以可以將所有的子樹重疊在一起，想像把它們壓縮成單一棵子樹，然後改變子樹中節點的資料結構，把每一個節點想像成一個 list。

圖 3.8 為側面示意圖，每一條粗橫線代表一棵子樹，虛線框框表示一份 list。因為在每個時間點，只會需要用到少部分的子樹，所以 list 其實不需要被完整地建立，而是採取動態管理的方式，只將需要用到的子樹加入到 list 中（倘若全部建立的話，等於沒有縮減記憶體）；在此，甚至可以更進一步地使各個 list 彼此獨立，每個 list 各自加入需要用到的子樹節點，因此部分的 list 甚至是空的；此外，這樣的資料結構使記憶體的管理更簡單清楚，而且擴充至 N 連語言模型也相當容易。

在實作上使用 C++ Map [10]（等同於 Associative array、Dictionary 或 Lookup table）作為上述 list 的容器，利用其快速查找的特性，可以很快地判斷是否該加入新的子樹節點，以達到快速管理記憶體的目的；使用 Map 也能夠使維特比演算法有效率地運作。

第四章、語言模型預查

在第三章中提到，想提早讓語言模型的資訊發揮作用的話，可以使用語言模型預查的方法，本章將詳細介紹語言模型的概念與詳細作法，說明單連與 N 連語言模型預查的差別，並介紹語言模型預查平滑化的方法。需要先說明的是，語言模型的機率會先取對數值之後再加到 hypothesis 上，本文中所指的語言模型分數，即為此對數值。

4.1 語言模型預查

使用語言模型預查的目的，就是希望能夠儘早加入語言模型的資訊，圖 4.1 是詞典樹的示意圖，每一個圓圈代表一個字元，其中雙層圓圈表示詞尾；原本語言模型的分數必須在離開詞尾時才能加入，如果採取語言模型預查的方式，語言模型分數會在詞轉移或字元轉移時被分批加入；例如“交通大學”一詞，原本在離開詞尾“學”時，才會加上語言模型的分數，如果改作語言模型預查，在進入詞首“交”時，便會給予部分的語言模型分數，之後每進到下一個字元會再給予部分的分數，在最後離開詞尾時，分批得到的語言模型分數，會等於原本一次全給所得到的分數。

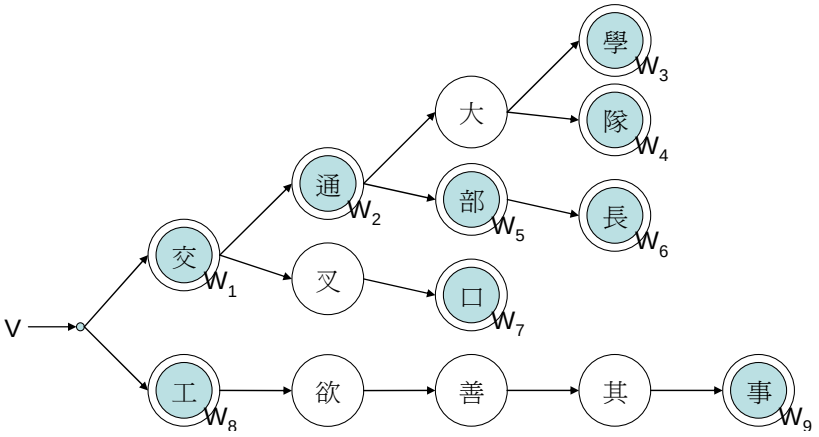

圖 4.1 詞典樹示意圖

它的基本概念很直覺，使 hypothesis 在進到每個字元 *c* 時所“累積”得到的

語言模型機率，等於以c為根節點的子樹內的所有詞彙當中，語言模型機率最大者，定義該值為$\pi_{c,V}$，

$$\pi_{c,V} = \max_{W_i \in S(c)} \{P(W_i \mid V)\} \tag{4.1}$$

其中下標中的V為其前詞，$S(c)$為一個以c為根節點的子樹內的詞彙所組成的集合。因此，預計在每個節點加入的語言模型機率$P_{c,V}$可以用下式表示，

$$P_{c,V} = \pi_{c,V} - \pi_{\bar{c},V} \tag{4.2}$$

其中$\bar{c}$為c的父節點字元。

根據上述的想法，在進入字元節點c時，系統會將$P_{c,V}$的對數值加到hypothesis上，此時該hypothesis上所累積的語言模型機率等於$\pi_{c,V}$（不是指全部累積的機率$P(W_1^n)$，而是只看$P(W_i \mid W_{i-N+1}^{i-1})$的部分）；到這個時間為止，系統提供了$S(c)$存活的最大可能性給進入$c$中的hypothesis，如果這個hypothesis仍然會被刪除演算法所剔除的話，表示$S(c)$中所包含的詞，其語言模型分數皆太低；反之，hypothesis順利留下的話，表示$S(c)$中至少有一個詞，其語言模型分數足以保護該hypothesis不被剔除。

假設圖4.1中，$P(W_{i+1} \mid V) > P(W_i \mid V)$，$i = 1 \sim 8$；當hypothesis進到字元“通”時，其累積獲得的語言模型機率等於$P(W_6 \mid V)$，假如該hypothesis順利存活，當其欲以“交通”為前詞做詞轉移進入新的詞彙時，必須將語言模型機率修正回$P(W_2 \mid V)$，因此，每次詞轉移之前，必須乘上一筆修正機率$D_{c,V}$，

$$D_{c,V} = P(W_c \mid V) - \pi_{c,V} \tag{4.3}$$

W_c為字元c所在的詞彙，要注意的是，c必須是詞尾才會進行詞轉移。

作個簡單的整理，若系統以語言模型預查方式提供語言模型機率的話，在每進入一個字元的第一個狀態，會加上$\log(P_{c,V})$，當離開每個字元的最後一個狀

態時，有做詞轉移的新 hypothesis，必須加上修正分數 $\log\left(D_{c,V}\right)$。表 4.1 為產生 $P_{c,V}$ 與 $D_{c,V}$ 的演算法， $P_{c,V}$ 與 $D_{c,V}$ 分別簡寫為 P 與 D。

表 4.1 語言模型預查演算法

```
predecessor   V
node          parent, root
array         Score        // store language model score
array         Max          // store maximum score in sub-tree
array         lmlaScore    // store  P_{c,V}  and  D_{c,V}

getLMLAScore ( V , lmlaScore )
{
    Score = fillScoreIntoLexicon ( V ) ;
    Max = getMaximumScore ( Score ) ;

    for all nodes 'n' except root
    {
        parent = parent node of n ;

        if ( parent != root ) lmlaScore.P[n] = Max[n] - Max[parent] ;
        else                  lmlaScore.P[n] = Max[n];

        lmlaScore.D[n] = lmlaScore.P[n] - Max[n] ;
    }
}
```

當前詞不同時，各詞的語言模型機率 $P\left(W_i \mid W_{i-N+1}^{i-1}\right)$將隨之改變，必須重新填寫 *Score*，再由此求 *Max*，有了 *Score* 與 *Max* 以後，便很容易求得 $P_{c,V}$ 與 $D_{c,V}$ 。

4.2 單連語言模型預查

在單連語言模型中，每個詞的機率只跟自己本身出現的機率有關，沒有前詞的問題，因此，不論 V 為何，*fillScoreIntoLexicon* 的結果都一樣，不會受到影響，如此一來，$P_{c,V}$ 與 $D_{c,V}$ 也都不會變；基於這個特性，可以在辨識前先行產生 $P_{c,V}$

與 $D_{c,V}$ ，辨識時便不需要重複同樣的運算。若是 N 連語言模型，每當前詞不同時，$P_{c,V}$ 與 $D_{c,V}$ 便會隨之改變，所以必須得在系統運作過程中，隨時產生需要用到的語言模型預查分數。

在實作上，可以使用 N 連語言模型，但只作單連語言模型預查，只要在詞轉移時再乘上另外一筆修正機率 $\tilde{D}_{c,V}$ 即可，

$$\tilde{D}_{c,V} = P\left(W_i \mid W_{i-N+1}^{i-1}\right) - P\left(W_i\right) \tag{4.4}$$

因為使用單連語言模型預查，所以在詞轉移前所得到的語言模型等於 $P\left(W_i\right)$，乘上 $\tilde{D}_{c,V}$ 後即可修正回 $P\left(W_i \mid W_{i-N+1}^{i-1}\right)$。這樣的作法與不做語言模型預查相比，增加的運算微乎其微，但辨識率與速度的提升有很好的效果，在第五章中將有更進一步的比較與分析。

4.3 雙連語言模型預查

使用 N 連語言模型，但只作單連語言模型預查時，如果再配合使用 3.2.4 節提到的詞轉移刪除法，可能會造成嚴重的問題；當詞轉移刪除法的門檻值設太高時，某些在單連語言模型預查時分數較低的詞將永遠不會出現，以圖 4.1 為例，仍舊假設 $P\left(W_{i+1} \mid V\right) > P\left(W_i \mid V\right)$，$i = 1 \sim 8$，當詞轉移後分別有進入“交”與進入“工”的 hypothesis 產生，進入前者的 hypothesis 獲得的機率為 $P\left(W_7\right)$，後者則為 $P\left(W_9\right)$，如果門檻值剛好被設在兩者之間，那麼進入前者的 hypothesis 將被詞轉移刪除法的機制所剔除，如果門檻值設得更高的話，可能兩者同時都會被剔除，其下所包含的詞也就不可能會出現在辨識結果中，這樣的作法，等於事前先將詞典經過篩選，只留下部分的詞彙；反之，當門檻值過低時，留下過多的 hypothesis，詞轉移刪除法無法發揮預期的效果。

表 4.2 語言模型預查之查表法

```
queue       order
array       lmlaScore     // store  P_{c,V}  and  D_{c,V}
2-D array   buffer        // composed of a few lmlaScore array

accessTable ( V , lmlaScore )
{
    if ( V is NOT in buffer )
    {
        if ( buffer is full )
        {
            erase oldest one in buffer ;
            order.pop() ;
        }

        order.push( V ) ;
        getLMLAScore ( V , lmlaScore ) ;
        buffer[V] = lmlaScore ;
    }
    else
        lmlaScore = buffer[V] ;
}
```

做 N 連語言模型預查時，每當前詞不同就必須重作 *getLMLAScore* 的動作，如果將每次得到的所有 $P_{c,V}$ 和 $D_{c,V}$ 暫時儲存起來 [8]，下次出現相同的前詞時，便可直接查表得到所需要的 $P_{c,V}$ 和 $D_{c,V}$ ，如果持續無限制地儲存，記憶體很容易就會耗盡，因此一般會設定儲存的上限，當儲存的筆數到達上限時，便移除一筆舊的資料，表 4.2 即為此查表法之演算法。

當需要移除舊資料時，採取最簡單的方式，直接剔除掉最舊的一筆資料。此外，當緩衝區的上限訂得高時，記憶體就會用得多，但是重作 *getLMLAScore* 的機會將變得很少，如果上限越低，便越容易遇到重複計算相同前詞 *getLMLAScore* 的機會。

使用 N 連語言模型預查再配合語言模型刪除法時，便不會有只作單連語言模型時的問題，能夠確保語言模型在辨識過程中發揮正確的功能。

4.4 語言模型預查平滑化

在圖 4.1 中，W_8 的語言模型機率為 $P(W_8|V)$，W_9 的語言模型機率為 $P(W_9|V)$，假設 $P(W_8|V) < P(W_9|V)$，可以得到圖 4.2a 的語言模型預查機率值，其中在“工”時就會得到 $P(W_9|V)$ 的機率。

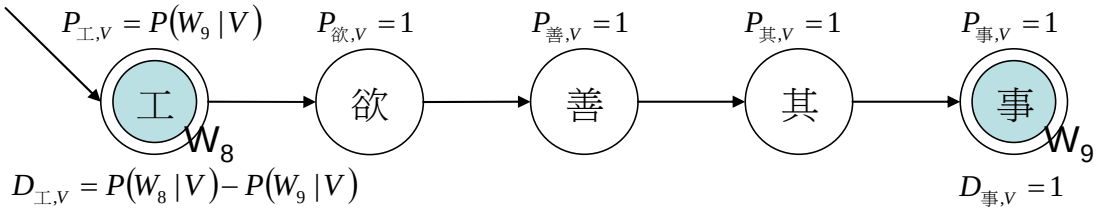

圖 4.2a 語言模型預查分數（未平滑化）

在大多數的情況下，語言模型的分數為一負數值（機率的對數值），因此，相較於分次將分數加到 hypothesis，如果一次就加上全部的分數，該 hypothesis 被聲學刪除法剔除的機會反而更大，圖 4.2b 便是將語言模型預查的分數分次加到 hypothesis 上，此即為語言模型預查平滑化 [11] 的概念。平滑化的作法能夠使 hypothesis 上的分數變化更為均勻，在每個字元上都能獲得部分的語言模型分數，取代了原本時加時不加（加的值為零，因為 $\log(1) = 0$）的情形，因為變化得更為均勻，所以可以使聲學刪除法的 Beam Width 縮得更小，有助於正確地剔除更多低分的 hypothesis。

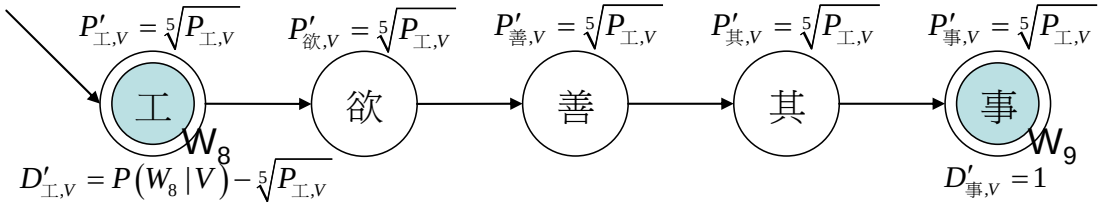

圖 4.2b 語言模型預查分數（平滑化）

詞典樹會在辨識開始前就先建構好，因此可以將需要做平滑化的節點預先標記起來，圖 4.3 中黑色實心節點就是需要做平滑化的節點，其分數將被平均分配至其下的斜紋節點；需要平滑化的節點具有一個簡單的特性，就是它只有一個

子節點，若其子節點仍只有一個子節點的話，它的分數將被平均分配到這三個節點上，其他相似的情況可依此類推。

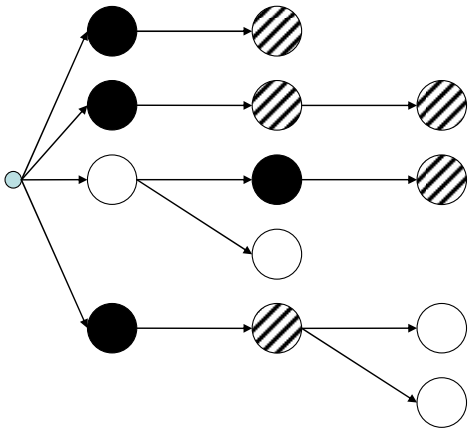

圖 4.3 需要作平滑化的節點

本系統的作法略有不同，儲存的不是黑色實心節點，而是最深一層的斜紋節點，並儲存需平滑化的層數，之後再依表 4.3 的演算法求平滑化後的 $P'_{c,V}$ 與 $D'_{c,V}$。

表 4.3 語言模型預查平滑化

```
runSmooth ( lmlaScore )
{
    for all nodes 'n' needed to be smoothed
    {
        top is the highest node ;
        low is the deepest node ;
        level is the depth from top to low ;
        float unit = top.P / level ;

        top.P = unit ;
        top.D += unit * ( level – 1 ) ;
        low.P = unit ;

        if ( level > 2 )
        {
            for ( i = 2 ; i < level ; i++)
            {
                node.P += unit ;
                node.D += unit * ( i – 1 ) ;
            }
        }
    }
}
```

延伸語言模型預查平滑化的概念，可以把 $P'_{c,V}$ 進一步平均分配到音節的每個狀態 [11]，以本系統為例，就是將 $P'_{c,V}$ 分成 8 份，每當進到下一個狀態，便加上 $1/8$ 的分數（以機率而言，為乘上 $P^{1/8}$），若是回到原本的狀態則不加語言模型相關分數（該狀態已經加過一次，不能重複加）；原本的作法，會在進入第一個狀態時加上 $\log\left(P'_{c,V}\right)$，之後的狀態不會獲得語言模型相關的分數，若是改作狀態的平滑化，那麼每個狀態都能獲得一小部分的分數，分數的變化會更為均勻，可以使用更小的寬度進行光束搜尋法，在第五章中將有相關的測試數據比較各種平滑化的效果。

第五章、實驗與分析

實驗分成兩個階段，第一個階段使用 Treebank 做為測試語料，其中又可再細分為幾個步驟，首先拿本系統的最佳辨識率與 HTK 的最佳辨識率比較，並且比較其辨識時間；再針對本系統的各項參數設計實驗，觀察它們對辨識率與辨識時間的影響；第二階段改使用 TCC300 中成功大學所錄製的部分音檔作為測試語料，同樣會跟 HTK 比較辨識結果，本階段主要是想測試系統對於非特定語者之辨識能力。

第一階段中 Treebank 的 HMM 為本實驗室先前所訓練，每個模型的 mixture 數量並不固定，而是依照資料的需要而有所不同，最高的數量為 101 mixtures，平均為 66.81 個，short pause 則為 129 mixtures。第二階段 TCC300 的 HMM 則是用 HTK 重新訓練過，固定將 mixture 的數量升至 32 個，short pause 則為 64 mixtures。

所有實驗使用相同詞典，詞彙數量皆為六萬詞，辨識時間，皆在同一台電腦上做測試，包含 HTK 的辨識時間也是在由該電腦所量測，CPU 型號為 Intel® Core™2 Extreme X9650，3.00GHz，作業系統為 Microsoft® Windows Server™ 2003 Standard Edition SP2。

5.1 Treebank

Treebank 的前 380 個音檔是訓練語料，從編號 397 號的音檔開始，選取相差 13 號的音檔作為測試語料，共有 60 個音檔，總音長為 1574.71 秒，共計有 5681 個字元，組成 3109 個詞。

本實驗的比較基準為 HTK 裡 HVite 函數之辨識結果，兩邊使用相同的聲學模型與語言模型。

5.1.1 HTK

本實驗的比較基準為 HVite 函數，先使用下式的命令產生*.lat 檔（Lattice file 或稱為 Word graph file），

HVite …… -l lattice -z lat -n 10 1 -t 300 -s 11.0 ……

之後再以 HLRescore 函數找出最佳的語言模型權重值，執行結果如表 5.1。從表中可以清楚地觀察到，語言模型的權重值對 HVite 的辨識結果有相當的影響，使用不當的權重值，可能會讓辨識率下降超過 5%。

表 5.1 不同 Weight 對辨識率的影響（HTK）

Weight	Acc(%)	
	syllable	character
0	88.00	55.96
1	89.56	77.06
2	90.81	79.07
3	91.81	80.62
4	92.48	81.85
5	92.99	82.86
6	93.24	83.30
7	93.43	83.56
8	93.72	83.98
9	93.94	84.49
10	93.91	84.63
11	94.07	84.81
12	94.10	84.84
13	94.17	85.11
14	94.21	85.16
15	94.14	85.11
16	94.03	85.00
17	94.07	85.11
18	93.96	85.16
19	93.72	84.97
20	93.45	84.63

將語言模型的權重值設為 14.0，使用不同寬度的 BeamWidth（HTK 的-t 參

數），測試其辨識率，指令為，

HVite …… -t BeamWidth -s 14.0 ……

關閉-p（Word insertion log probability）或-v（Word end pruning）等參數，量測其辨識時間（Run-time）；RTF（Real Time Factor）則為辨識時間除以總音長之商數。

表 5.2 為其結果，可以觀察到，當 BeamWidth 越大時，辨識率越高，其原因在於每個時間點所保留的 token 越多的話，演算法越接近全域搜尋（Full search），辨識率也就越接近最佳解；此外，當 BeamWidth 大於 325 後，辨識率不再提升，這點也間接地為 3.2.4 節中，光束搜尋演算法的可行性提出佐證。

表 5.2 不同 BeamWidth 對辨識結果的影響（HTK）

BeamWidth	Acc(%)		Run-time(s)	RTF
	syllable	character		
25	0.00	0.00	75.968	0.048
50	0.00	0.00	80.516	0.051
75	2.94	1.81	27.531	0.017
100	14.78	8.34	59.765	0.038
125	46.26	31.83	161.516	0.103
150	78.47	61.90	423.953	0.269
175	88.58	76.11	1036.030	0.658
200	90.62	80.65	2453.360	1.558
225	93.50	84.30	4704.380	2.987
250	93.61	84.44	6553.160	4.162
275	94.23	85.11	8040.550	5.106
300	94.23	85.14	9343.250	5.933
325	94.21	85.16	10488.300	6.660
350	94.21	85.16	12411.800	7.882
375	94.21	85.16	14167.700	8.997
400	94.21	85.16	15938.900	10.122

由表 5.2 可以知道，HTK 字元的最佳辨識結果為 85.16%，需要 6.66 倍的實際時間；但當 BeamWidth 設定為 225 時，辨識結果已經相當接近最佳結果（小於 1% 視為接近），只需要 2.987 倍的實際時間。

5.1.2 辨識系統

除了語言模型權重值之外，先將其他參數固定，找到適當的權重值之後，再針對不同寬度的 BeamWidth，以及不同數量的 WordEnd hypotheses（關係到每個時間點，允許做詞轉移的 hypothesis 的最大數量）作測試，並開啟語言模型預查及狀態平滑化，刪除演算法則只用聲學刪除法，而不作統計式刪除法及詞轉移刪除法，以期能找出最佳辨識率及辨識時間。

首先固定 BeamWidth = 150.0，WordEnd = 3 個，變化不同大小的權重值（Weight），實驗結果如表 5.3。在 Weight = 13.0 時，可以得到最好的字元辨識率 84.30%。之後的測試，將選定 Weight = 13.0 當作語言模型的權重值。

表 5.3 不同 Weight 對辨識率的影響

Weight	Acc(%)		Run-time(s)
	syllable	character	
7	93.45	83.26	4055.506
8	-	-	-
9	93.72	83.86	3700.583
10	-	-	-
11	92.85	83.28	3666.921
12	93.68	84.16	3358.793
13	93.70	84.30	3223.046
14	93.73	84.28	3059.440
15	93.61	84.07	3008.764
16	93.40	84.02	2956.062
17	93.21	83.65	2653.856

固定 Weight = 13.0，WordEnd = 3 個，觀察不同 BeamWidth 對辨識結果的影響，結果在表 5.4。可以發現 BeamWidth 對辨識率有很大的影響，當 BeamWidth = 250 時，有最高的辨識率 84.69%，當 BeamWidth 超過 150 以後便有相當接近的辨識率（同樣以小於 1%視為接近），與 HTK 有著類似的現象，當 BeamWidth 足夠大時，辨識率便會開始呈現穩定狀態，由此可以驗證在本系統中使用詞轉移

刪除法的可行性。

同樣固定 Weight = 13.0，使用較多的 WordEnd = 6 個，觀察不同 BeamWidth 對辨識率與辨識時間的影響，結果如表 5.5。當 BeamWidth 高於 150 後，辨識率便開始穩定；在 BeamWidth = 200.0 時有最佳辨識率 85.23%，略高於 HTK 的 85.16%，時間上只要 3.525 倍的實際時間，相較於 HTK 的 6.66 倍，少了 47.1% 的時間。

表 5.4 不同 BeamWidth 對辨識結果的影響（WordEnd = 3）

BeamWidth	Acc(%)		Run-time(s)	RTF
	syllable	character		
50	44.15	32.02	278.854	0.177
75	80.67	67.59	755.820	0.480
100	92.55	82.13	1422.552	0.903
125	93.42	83.81	2035.222	1.292
150	93.70	84.30	2375.876	1.509
175	93.87	84.56	3088.538	1.961
200	93.86	84.58	3813.381	2.422
225	-	-	-	-
250	93.89	84.69	5431.214	3.449

表 5.5 不同 BeamWidth 對辨識結果的影響（WordEnd = 6）

BeamWidth	Acc(%)		Run-time(s)	RTF
	syllable	character		
50	44.27	32.21	266.555	0.169
75	80.92	68.54	657.869	0.418
100	92.82	82.87	1446.885	0.919
125	93.70	84.12	2347.450	1.491
150	93.96	84.86	3488.972	2.216
175	94.23	85.16	4611.714	2.929
200	94.17	85.23	5550.145	3.525
225	-	-	-	-
250	94.17	85.09	9123.701	5.794

接著同樣固定 Weight = 13.0，分別使用 BeamWidth = 150.0 與 BeamWidth = 200.0，嘗試在每個時間點中留下不同數量的 WordEnd 做詞轉移的動作，系統會先將可以做詞轉移的 hypothesis 排序，只讓其中分數最高的幾個做轉移，測試的結果分別在表 5.6 跟表 5.7。當 WordEnd 數量等於 6 時，有最好的辨識效果，分別是 84.86%、85.23%。事實上，在兩種不同的條件下，當 WordEnd = 3 時，皆已經接近最好的辨識結果，而且只需要 1.509 倍或 2.422 倍的實際時間，由此可知，每個時間點留下 3 個以上的候選詞時，候選詞的涵蓋率已經相當地高。

表 5.6 不同 WordEnd 對辨識結果的影響（BeamWidth = 150.0）

WordEnd	Acc(%)		Run-time(s)	RTF
	syllable	character		
1	93.50	83.47	1845.296	1.172
2	93.50	83.47	1859.411	1.181
3	93.70	84.30	2375.876	1.509
4	93.87	84.44	2689.664	1.708
5	94.00	84.69	3214.707	2.041
6	93.96	84.86	3488.972	2.216
7	94.00	84.77	3603.371	2.288

表 5.7 不同 WordEnd 對辨識結果的影響（BeamWidth = 200.0）

WordEnd	Acc(%)		Run-time(s)	RTF
	syllable	character		
1	93.72	83.88	3058.697	1.942
2	93.72	83.88	3241.656	2.059
3	93.86	84.58	3813.381	2.422
4	93.96	84.65	4794.161	3.044
5	94.07	84.84	4831.003	3.068
6	94.17	85.23	5550.145	3.525
7	94.10	84.97	5907.387	3.751

Weight = 13.0，將不同 BeamWidth 與不同 WordEnd 條件下的字元辨識率整理成表 5.8，辨識時間整理在表 5.9。它的趨勢很明顯，越往表格的右下方，字元

辨識率越高，辨識時間亦越長。

表 5.8 各種 BeamWidth 與 WordEnd 下的辨識率

Acc(%)	WordEnd					
BeamWidth	1	2	3	4	5	6
50	-	-	32.02	-	-	32.21
75	-	-	67.59	-	-	68.54
100	-	-	82.13	-	-	82.87
125	-	-	83.81	-	-	84.12
150	83.47	83.47	84.30	84.44	84.69	84.86
175	-	-	84.56	84.62	84.79	85.16
200	83.88	83.88	84.58	84.65	84.84	85.23

表 5.9 各種 BeamWidth 與 WordEnd 下的 RTF

RTF	WordEnd					
BeamWidth	1	2	3	4	5	6
50	-	-	0.177	-	-	0.169
75	-	-	0.480	-	-	0.418
100	-	-	0.903	-	-	0.919
125	-	-	1.292	-	-	1.491
150	1.172	1.181	1.509	1.708	2.041	2.216
175	-	-	1.961	2.457	3.028	2.929
200	1.942	2.059	2.422	3.044	3.068	3.525

5.1.3 詞轉移刪除法測試

以 Weight = 13.0，BeamWidth = 200.0，WordEnd = 6 個，雙連語言模型預查，作狀態平滑化，但不使用詞轉移刪除法的測試結果（Acc = 85.23%，RTF = 3.525）作為比較的基準，使用不同的詞轉移刪除法門檻值，觀察紀錄其對辨識的影響在表 5.10，最右邊欄的 Diff.指的是與原 RTF 的差距百分比。可以觀察到，測試語料中的詞絕大多數出現在門檻值 4.8 以內，其辨識時間可以減少超過 30%，其結果幾乎與不作詞轉移刪除法的辨識結果相當。

當門檻值高於 9.0 後，辨識率等於不作詞轉移刪除法，可以視同沒有候選詞

因此被剔除，然而辨識時間卻會因為執行詞轉移刪除法而拖慢，詳細分析其原因在於，每個可以進行詞轉移的 hypothesis 會產生四千多個 hypothesis，每個時間點可能會有數個 hypothesis 進行轉移，當執行了產生與否的判斷後，卻極少 hypothesis 因此不被產生，這樣一來反而浪費了更多時間，從這一點也可以發現到，雖然只是增加簡單的判斷，卻會對最後的辨識時間影響很大，可見得詞轉移的部分，占了系統相當大的運算量。

分別將字元辨識率及 RTF 對門檻值作圖，結果分別畫在圖 5.1 及圖 5.2，兩者呈現類似的趨勢，其中圖 5.1 上的虛線，橫向的虛線表示與最高辨識率差距 1%，其與趨勢線的交點約落在橫軸 4.8 附近，在一般用途的語音辨識系統上可以將門檻值設定在這附近。

表 5.10 詞轉移刪除法對辨識的影響

Threshold	Acc(%)		Run-time(s)	RTF	Diff. (%)
	syllable	character			
4.0	92.25	78.38	2723.156	1.729	-50.9
4.5	93.58	82.75	3413.996	2.168	-38.5
4.6	93.89	83.81	3525.545	2.239	-36.5
4.7	93.98	84.03	3687.411	2.342	-33.6
4.8	94.10	84.46	3807.631	2.418	-31.4
4.9	94.17	84.53	4478.261	2.844	-19.3
5.0	94.14	84.77	4478.673	2.844	-19.3
6.0	93.84	84.99	5717.489	3.631	3.0
7.0	94.23	85.18	6539.170	4.153	17.8
8.0	94.14	85.20	6423.613	4.079	15.7
9.0	94.17	85.23	6527.378	4.145	17.6
10.0	94.17	85.23	6401.620	4.065	15.3

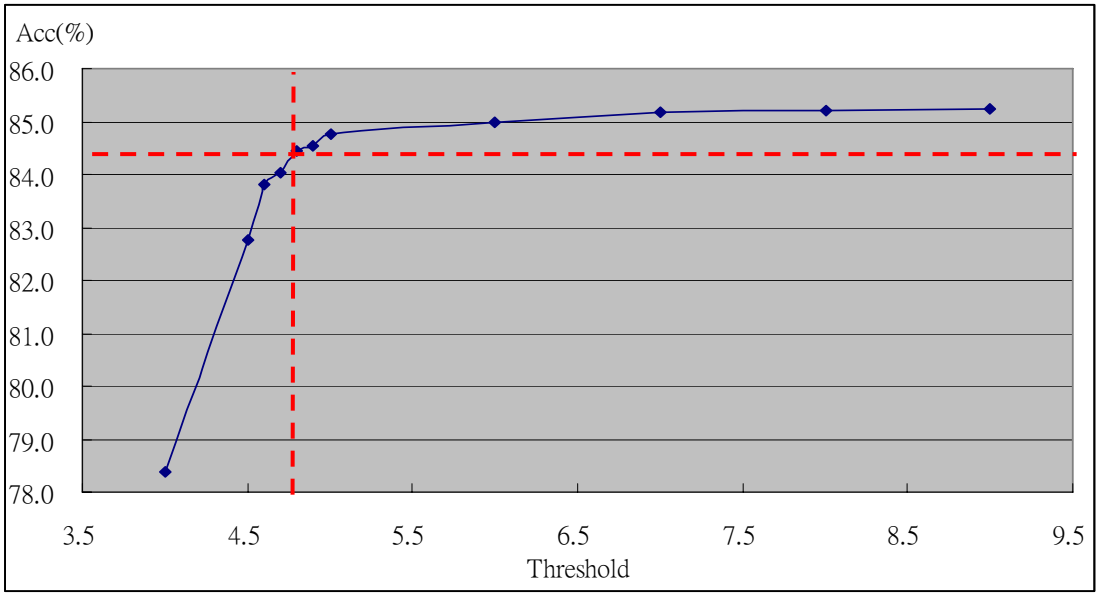

圖 5.1 各種詞轉移刪除法門檻值下的辨識率

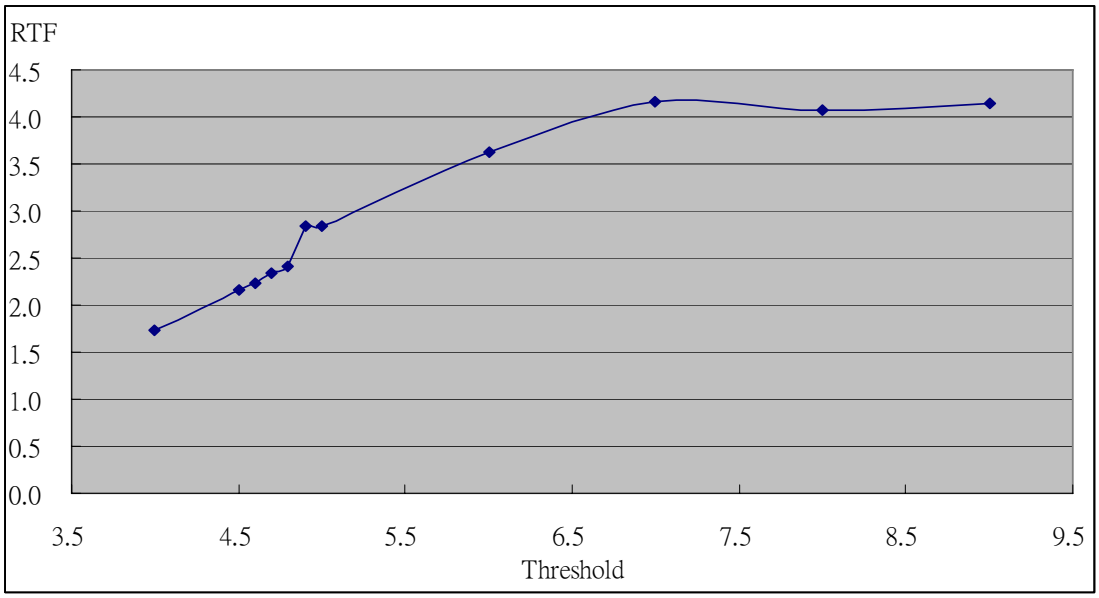

圖 5.2 各種詞轉移刪除法門檻值下的 RTF

5.1.4 語言模型預查平滑化測試

同樣以 Weight = 13.0，WordEnd = 6 個，雙連語言模型預查，作狀態平滑化，但不使用詞轉移刪除法的測試結果作為比較的基準（表 5.5），分別對只作字元平滑化（不平滑化至狀態）以及只作預查但不作平滑化兩種情形做試驗，觀察辨識率及辨識時間。

表 5.11 為只做字元平滑化的測試結果，在 BeamWidth 不大時，辨識效果較差，可以看出狀態平滑化對縮減 BeamWidth 的功效；加大 BeamWidth 之後，辨識率開始明顯提升，最好的辨識率則優於作狀態平滑化的結果，而且速度快了 57.7%，原因在於詞轉移時的運算量較少，語言模型的分數也不需要分很多次加，所以速度能夠提升。

表 5.11 字元平滑化的辨識結果

BeamWidth	Acc(%)		Run-time(s)	RTF
	syllable	character		
50	1.35	0.46	43.035	0.027
75	6.38	3.62	70.356	0.045
100	42.25	32.78	306.765	0.195
125	83.68	71.84	808.675	0.514
150	92.87	83.19	1194.693	0.759
175	94.02	84.77	1661.495	1.055
200	94.24	85.34	2349.297	1.492
250	94.19	85.20	4694.913	2.981

表 5.12 不作平滑化的辨識結果

BeamWidth	Acc(%)		Run-time(s)	RTF
	syllable	character		
50	13.22	7.75	162.404	0.103
75	53.95	35.93	319.859	0.203
100	85.20	67.91	548.629	0.348
125	92.94	82.56	792.332	0.503
150	93.91	84.81	1213.238	0.770
175	94.09	85.07	1605.700	1.020
200	94.19	85.39	2524.262	1.603
250	94.12	85.16	4460.558	2.833

表 5.12 為完全不作平滑化的測試結果，與只作字元平滑化的測試結果有類似的現象， BeamWidth 寬時辨識率高，且速度快。將三種設定的字元辨識率與 RTF 放在表 5.13 中一起比較，發現作狀態平滑化，當 BeamWidth 很窄時，就能

有相當高的辨識率；而不作任何的平滑化的話，在 BeamWidth = 200.0 時，不但可以有最好的辨識率，其辨識時間亦相當地快，比作狀態平滑化時少了 54.4%，跟只作字元平滑化相比的話，雖然速度稍慢，但在相同條件下，不作平滑化的辨識率仍然較高，特別是在 BeamWidth 較窄時，差距更明顯，推測其原因，應該是中文語音辨識裡，語言模型的影響超乎原本預期，在詞轉移或字元轉移時，加重語言模型的影響力，也許將有助於辨識率的提升，而平滑化的動作卻是反將影響力平均化，未來可以再作進一步的實驗加以驗證。

表 5.13 綜合比較平滑化的效果

BeamWidth	Smooth to 'State'		Smooth to 'Character'		No Smooth	
	Acc(%)	RTF	Acc(%)	RTF	Acc(%)	RTF
50	32.21	0.169	1.35	0.460	7.75	0.103
75	68.54	0.418	6.38	3.620	35.93	0.203
100	82.87	0.919	32.78	0.195	67.91	0.348
125	84.12	1.491	71.84	0.514	82.56	0.503
150	84.86	2.216	83.19	0.759	84.81	0.770
175	85.16	2.929	84.77	1.055	85.07	1.020
200	85.23	3.525	85.34	1.492	85.39	1.603
250	85.09	5.794	85.20	2.981	85.16	2.833

5.1.5 語言模型預查測試

接著以 Weight = 13.0，WordEnd = 6 個，雙連語言模型預查，不作平滑化，不使用詞轉移刪除法的測試結果（表 5.12）作為比較的基準，與（使用雙連語言模型）只作單連語言模型預查，以及（使用雙連語言模型）完全不作語言模型預查的系統，比較其辨識率，因為程式流程並未針對兩者作最佳化，故不比較辨識時間。最後將三者的辨識結果一起列在表 5.14 中比較。

可以清楚地觀察到，單連語言模型預查的辨識率都比雙連語言模型預查略低；至於不做語言模型預查，其音節辨識率雖仍有一定水準，但字元辨識率極低，也許將 BeamWidth 設得更寬會有所幫助，不過因為辨識時間已經很長，而且辨

識率仍相當低，因此不再往下測試。

表 5.14 綜合比較語言模型預查的效果

BeamWidth	2-gram LMLA		1-gram LMLA		No LMLA	
	syllable	character	syllable	character	syllable	character
50	13.22	7.75	3.68	2.29	2.31	1.51
75	53.95	35.93	40.77	24.43	2.83	1.87
100	85.20	67.91	82.03	63.63	5.00	3.70
125	92.94	82.56	92.36	79.74	10.07	8.08
150	93.91	84.81	93.84	83.30	45.68	32.32
175	94.09	85.07	93.93	83.47	85.69	49.69
200	94.19	85.39	93.87	83.72	88.26	45.34
250	94.12	85.16	93.91	83.89	88.03	45.12

5.1.6 記憶體使用量

關於記憶體的使用量，作以下簡單的測試，首先，挑選測試語料當中最長的一個音檔，長度為 33.92 秒，總共有 112 個字元，然後觀察系統在辨識該音檔的過程中，記憶體的最大使用量；因為是最長的音檔，所以在辨識過程中需要的記憶體，應該足以代表本系統的記憶體需求量。

在 3.3.2 節中提到，使用語言模型預查需要更多的記憶體，從第四章可以知道這些記憶體是用來當緩衝區，暫時儲存語言模型預查的分數；以下使用不同大小的緩衝區，觀察其對記憶體的影響，並測試它對辨識時間的影響。其結果在表 5.15。

系統的設定為 Weight = 13.0，BeamWidth = 200.0，WordEnd = 6 個，使用雙連語言模型，並作預查，但不作平滑化，不使用詞轉移刪除法。當緩衝區越大時，語言模型預查重新計算分數的機會便降低，結果正如表 5.15 的 RTF 一項所示，當緩衝區越大，辨識速度越快；相對地，緩衝區越大，代表需要用到更多的記憶體，保留 1,000 筆資料需要比 100 筆多約 500MB 的記憶體，然而對辨識速度的影響已經很小，因此，本章中其他項目的設定皆為保留 100 筆資料。

表 5.15 記憶體使用量對 RTF 的影響

Num	KB	RTF
10	396,880	3.438
20	402,440	1.459
30	407,848	1.324
40	413,580	1.301
50	419,116	1.297
100	447,388	1.280
1000	952,808	1.241

若是進一步觀察可以發現，除非緩衝區設定過小，否則對辨識速度的影響，皆在 5%以內，以本測試音檔為例，其在每個時間點存活下的 hypotheses，具有各自不同的前詞，每個時間點中，前詞的平均值數量為 12.71 個，大部分的前詞都會持續存在一小段時間，因此當設定過小時，有些相同的預查分數，就會被連續重複計算好幾次，造成辨識速度減慢。

5.2 TCC300

TCC300 測試音檔的名單列在附錄中，共計有十個語者（男女各半），2349 個字元，總音長為 646.58 秒。

表 5.16 不同 Weight 對辨識率的影響（HTK）

Weight	Acc(%)	
	syllable	character
8	84.16	75.44
9	84.16	75.69
10	83.91	75.65
11	83.87	75.90
12	83.10	75.35
13	83.06	75.31
14	82.80	75.10
15	81.82	73.95

同樣先找出 HTK 的最佳辨識率及其 RTF。首先產生 Lattice file，找出最佳的語言模型權重值（表 5.16）；然後使用不同的 BeamWidth，觀察其辨識率與 RTF（表 5.17）。可以得到以下的結果，當 BeamWidth ≥ 275.0 時，HTK 可以得到最佳字元辨識率 75.90%，需要 18.789 倍的實際時間。

表 5.17 不同 BeamWidth 對辨識結果的影響（HTK）

BeamWidth	Acc(%)		Run-time(s)	RTF
	syllable	character		
50	3.13	2.47	14.171	0.022
75	15.86	7.80	31.265	0.048
100	52.02	36.87	84.047	0.130
125	69.31	57.47	253.750	0.392
150	78.16	68.41	957.453	1.481
175	83.31	74.67	2949.580	4.562
200	83.57	75.05	4867.980	7.529
225	83.87	75.78	6995.030	10.819
250	83.91	75.86	8979.530	13.888
275	83.87	75.90	12148.500	18.789

接著對系統作測試，首先找出最適當的語言模型權重值，設定 BeamWidth = 150.0，WordEnd = 3 個，觀察不同 Weight 下的辨識率，結果在表 5.18，當 Weight = 9.0 時的辨識率最好。

表 5.18 不同 Weight 對辨識率的影響

Weight	Acc(%)	
	syllable	character
7	82.38	72.71
8	82.84	73.73
9	83.18	74.20
10	82.93	73.90
11	82.72	73.39

接著固定 Weight = 9.0，分別對 BeamWidth 與 WordEnd 做測試，結果分別在表 5.19 與表 5.20。可以觀察到，本系統最好的字元辨識率為 74.33%，雖然不及 HTK 的 75.90%，但也相差不大，而且辨識的速度分別只要 HTK 的 23.4%與 46.3%。

表 5.19 不同 WordEnd 對辨識結果的影響（BeamWidth = 150.0）

WordEnd	Acc(%)		Run-time(s)	RTF
	syllable	character		
1	82.03	72.16	1385.730	2.143
2	82.03	72.16	1404.905	2.173
3	83.18	74.20	1891.793	2.926
4	83.14	73.82	2278.234	3.524
5	83.35	74.03	2635.266	4.076
6	83.44	74.33	2843.929	4.398

表 5.20 不同 BeamWidth 對辨識結果的影響（WordEnd = 3）

BeamWidth	Acc(%)		Run-time(s)	RTF
	syllable	character		
50	34.70	20.01	101.536	0.157
75	76.67	63.56	196.062	0.303
100	82.38	72.97	373.614	0.578
125	83.10	73.95	772.179	1.194
150	83.18	74.20	1891.793	2.926
175	83.23	74.29	3123.812	4.831
200	83.18	74.33	5618.704	8.690

第六章、結論與展望

語音辨識是一個牽涉廣泛的問題，可以試著從各個層面來提升辨識率、加快辨識速度，例如使用更強健的聲學模型，使用語言資訊更豐富的語言模型，使用更好的搜尋演算法等；在本章中，將把焦點單純放在本系統目前的架構，針對不足之處，提出可能可以改善的方向或概念，共可分為四點。

第一點，在語言模型預查中，當緩衝區滿了欲移除舊資料時，會直接將最舊的一筆資料剔除，若改成移除最久以前被用過的一筆資料，或是移除機率最低的資料等，其他更為有意義的方法，便能減少語言模型預查重複計算的次數，加快辨識速度，也能使緩衝區使用更有效率，進而縮減緩衝區的大小；然而要注意的是，判斷的方法不該過於耗時，否則反而會造成負擔。

第二點，中文是有聲調（Tone）之分的語言，但系統目前的作法沒有利用到這點，有一部分的辨識錯誤便是來自於相同音節但不同聲調的詞，間接使得語言模型的分數跟著加錯，若能將聲調的因素加到辨識中，應該能夠對提高辨識率有所幫助 [12]。聲調是一種超音段（Suprasegmental）的特徵，相對的系統用的是以時間對位為基礎的演算法，如何萃取出聲調的特徵，然後正確地加到演算法中，是首先要克服的困難點。

第三點，利用屬性偵測（Attribute detection）提早剔除錯誤的 hypothesis；hypothesis 的成長速度驚人，若能越早判斷出能否將它剔除，就能減輕許多的運算量，但是當屬性偵測不夠可靠，錯誤機率太高時，對辨識率會有很大的傷害；反之，若建立一套快速、可靠的屬性偵測的方法，就有機會能將語音辨識的速度大幅提升。

第四點，是針對記憶體的使用量，在辨識過程中，會將回溯的路徑儲存起來，藉此找回最佳存活路徑，仔細觀察發現這部分所耗去的記憶體，其實也占了不小的比例，特別是當辨識的語音長度很長的時候，會累積更多的回溯標記；但

實際上，有許多路徑沒有辦法存活到最後，因此可以每隔一段時間，對先前所儲存的回溯標記作整理，將不需要的標記刪除，如此一來，記憶體的最大使用量就能獲得控制。

語音辨識尚有許多值得探討的問題，如果想要讓這項技術融入一般人的生活中，還需要很多的努力，希望種種的困難能被早日克服，讓語音辨識成為廣泛應用的成熟技術，為人們的生活提供更多便利。

參考文獻

[1] Cambridge university Engineering Dept. (CUED), “Hidden Markov Model Toolkit (HTK)”, http://htk.eng.cam.ac.uk, version 3.4.

[2] S. Ortmanns, H. Ney, and A. Eiden, “Language-model look-ahead for large vocabulary speech recognition”, International Conference on Spoken Language Processsing, Philadelphia, PA, Oct. 1996, pp. 2095–2098.

[3] J. J. Wolf and W. A. Woods, “The HWIM speech understanding system,” in Trends in Speech Recognition, W. A. Lea, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1980, pp. 316–393.

[4] L. D. Erman and V. R. Lesser, “The hearsay-II speech understanding system,” in Trends in Speech Recognition, W. A. Lea, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1980, pp. 361–381.

[5] F. Jelinek, “Continuous speech recognition by statistical methods,” Proc. IEEE, Apr. 1976, vol. 64, pp. 532–556.

[6] S.J. Young, N.H. Russell and J.H.S. Thornton, “Token passing: a conceptual model for connected speech recognition systems,” CUED Technical Report F_INFENG/TR38, Cambridge University, 1989.

[7] Y.C. Pan, C.H. Yu and L.S. Lee, "Large vocabulary continuous Mandarin speech recognition using finite state machine," Chinese Spoken Language Processing, Dec. 2004, pp. 5-8.

[8] H. Ney and S. Ortmanns, “Progress in dynamic programming search for LVCSR,” Proceedings, IEEE, Aug. 2000, Vol. 88, pp. 1224 - 1240.

[9] 王小川, “語音訊號處理,” 全華科技圖書, Mar. 1994, chap. 12.

[10] The C++ Standards Committee, “ISO/IEC 14882 Programming Language C++,” 1998.

[11] D. Willett, E. McDermott and S. Katagiri, "Smoothed language model incorporation for efficient time-synchronous beam search decoding in LVCSR," Automatic Speech Recognition and Understanding, IEEE, 2001, pp. 178- 181.

[12] L. Xin and M. Ostendorf, "Word-Level Tone Modeling for Mandarin Speech Recognition," Acoustics, Speech and Signal Processing, IEEE, Apr. 2007, Vol. 4, pp. 665-668.

[13] H. Ney and X. Aubert, "A Word Graph Algorithm for Large Vocabulary, Continuous Speech Recognition," ICSLP, 1994.

[14] F. Wessel, R. Schluter, K. Macherey and H. Ney, "Confidence measures for large vocabulary continuous speechrecognition," Speech and Audio Processing, IEEE, Mar. 2001, Vol. 9, pp. 288-298.

[15] S. Ortmanns, A. Eiden, H. Ney and N. Coenen, "Look-ahead techniques for fast beam search," ICASSP, IEEE, 1997.

附錄

The list of the files tested in the section 5.2:

NCKU_f060602_0.mfc
NCKU_f060607_0.mfc
NCKU_f070303_0.mfc
NCKU_f070308_0.mfc
NCKU_f080104_0.mfc
NCKU_f080109_0.mfc
NCKU_f090405_0.mfc
NCKU_f090410_0.mfc
NCKU_f100301_0.mfc
NCKU_f100306_0.mfc
NCKU_m061005_0.mfc
NCKU_m061010_0.mfc
NCKU_m070101_0.mfc
NCKU_m070106_0.mfc
NCKU_m080903_0.mfc
NCKU_m080908_1.mfc
NCKU_m090902_0.mfc
NCKU_m090907_0.mfc
NCKU_m100904_0.mfc
NCKU_m100909_0.mfc