基於詞彙式大字符集、二階預測模型之文本資料壓縮

Text Compression Based on Word-Based Large Alphabet and Order-Two Prediction Model

古鴻炎　　溫智旻
Hung-Yan Gu and Chih-Ming Wen

國立台灣科技大學資訊工程系
e-mail: guhy@mail.ntust.edu.tw http://www.csie. ntust.edu.tw

摘要

本論文研究基於詞彙式大字符集之文本資料壓縮方法，將中、英文檔案以詞彙為單位剖析出token，再對 token 以二階混合式預測模型或部分匹配預測模型來估算出現機率，接著以算術編碼對該機率編碼。由於字符集很大，影響預測模型的處理速度，因此我們也研究一些可以加快處理的方法。

將上述想法實作為實際可壓縮、解壓縮的程式後，作壓縮率的測試實驗，再和 PPMd、bzip2、及 GZIP 程式作比較。對於中文檔案，我們的平均壓縮率，比 PPMd 好 1.12%，比 bzip2 好 5.48%，比 GZIP 好 17.02%。對於英文檔案，平均壓縮率則比 PPMd 好 0.29%，比 bzip2 好 2.04%，比 GZIP 好 12.08%。所以，文本資料壓縮率的改進，相當不容易，而本研究或多或少都得到了一些改進。

關鍵詞: 資料壓縮，大字符集，算術編碼

1. 前言

在此資訊愈來愈發達的時代，電腦及網路幾乎已經成為每個家庭必備的工具。而資料壓縮最主要的功用，就是可以節省電腦的儲存空間，也可以在有限頻寬的網路環境下，減少檔案傳輸所花的時間。

如今已有許多書籍內容都被轉換成可以用電腦來儲存及處理的格式。如：圖書館中的書本資料，研究論文放在網上供人瀏覽、查詢，技術文件、軟體使用說明隨產品附在光碟片中供使用者閱讀等。由於這些文本(text)資料愈來愈普遍，為了讓電腦中的文本資料，以更有效率的方式作儲存與傳輸，研究文本資料的壓縮技術是必要的。

1.1 相關研究

在文本(text)資料壓縮的研究領域，我們以三個處理步驟來討論過去被提出的方法。處理步驟為：剖析、模式(modeling)、編碼(coding)。

剖析(parsing): 剖析步驟主要是決定輸入資料裡每次要處理的符號單位的大小，將取出的"符號單位"(token)送到模式組件去估算機率。

對英文文章剖析的相關研究，如文獻[13]中將大字符集的觀念應用到一些常用的壓縮演算法上面，將英文文章的組成分類成字母、數字字串(alphanumeric string)與標點符號字串(punctuation string)兩類，因此一篇文章看成是不等長度之兩類字串所串成，並且把每個字串當作一個 token，此外在他們的研究中，也嘗試將英文詞彙分詞性來增加預測的準確性。後來 Moffat、Neal、Witten 在 1995 年提出的論文中[8]，對算術編碼作一些配合大字符集處理的修改，他們的剖析處理與文獻[13]裡的相同，並限制每個字串的最大長度為 16 個字元。另外在 2005 年由 Gu 提出的論文中[10]，對英文文章採用類似中文文章的剖析方法，依照一些判斷規則，一次取一個或兩個 byte 作為 token。

在中文文章剖析方面，論文[9]中有提到許多過去中文壓縮相關的研究，而他們自己的研究，對中文文章一次取兩個 byte 為 token，實驗結果顯示壓縮效果比取一個 byte 為 token 還好。在論文[14]中，將大字符集觀念應用在詞典式編碼上，以中文字元為 token，利用樣式匹配(pattern matching)的方式將數個連續的中文字元取成一個中文詞彙放在詞典中。此外論文[10, 15]中對中文文章的剖析，則是將連續兩個byte滿足BIG5碼定義範圍的條件下[16]，以該兩個 byte 為 token。

模式(modeling): 模式步驟主要是建立模型來估計各個字符集符號的出現機率，然後再將這個機率分布送給編碼模組作參考。在模式步驟所用的模型若是不同，則各個符號被估算出的機率也會不同。資料壓縮處理所使用的機率預測模型，可依預測的方式，分成靜態(static)模型、半調適(semi-adaptive)模型、及調適(adaptive)模型[6]。在靜態模型與半調適模型裡，機率分布一直維持不變，且半調適模型為了得到文本檔案中所有符號的機率分布，必須作 two-pass 的壓縮處理。為了作 one-pass 的壓縮處理，及動態掌握文章特性的變化，以下我們只討論、研究調適式的模型。

與模式相關的研究大致上可分為兩類：詞典式模型(dictionary-based models)與預測式模型(predictive models)。詞典式模型是將一個檔案前面

壓縮過的文句，建立一個詞典模型，之後如果遇到詞典中的符號，則以它在詞典中的索引表示。與詞典式模型相關的研究，如在 1977、1978 年由 Ziv 和 Lempel 發表的 LZ77、LZ78 壓縮法[3, 4]，以及 1984 年由 Welch 對 LZ78 改進之 LZW 壓縮法[5]，及其它的改進作法。

預測式模型是利用壓縮過的符號來預測目前可能遇到的符號的出現機率。但如果在最近幾個符號出現的條件下(即估算條件機率)，目前要壓縮之符號沒出現過，就會遇到所謂的零頻(zero frequency)問題，也就是目前符號的機率為零。解決的方法是利用一個特殊符號「逃脫符號(escape symbol)」來表示機率為零的情況，而逃脫符號的出現機率設定，在過去的文獻中可以找到相關研究[6, 7, 8, 9, 10]。預測式模型最具代表性的是由 Cleary 及 Witten 在 1984 年提出之"部分匹配預測模型"(prediction by partial match, PPM)[11]，基於此方法再配上算術編碼，可以達到非常好的壓縮效果，因此後來不少研究者對於部分匹配預測模型的逃脫機率估算方面，再作改進及發展出一些壓縮效果更好的演算法，如：PPMC、PPMD、PPMZ、PPMII 等[12]。本論文在模式步驟也採用了部分匹配的預測模型的方法。

編碼(coding): 編碼步驟就是將一個符號依照其出現機率決定要編的碼長及碼，出現機率大的符號使用較短的碼來編該符號，如此便可減少整個文本資料編碼後的檔案長度。

基於這個觀念，在 1952 年由 Huffman 提出的 Huffman 編碼[1]，他將每個符號依出現機率用整數個位元來編碼，該編碼方法的優點是編碼速度快，應用廣泛，例如 Win-RAR、JPEG 等都使用了 Huffman 編碼。在 1976 年，Rissanen 提出算術編碼[2]，它和 Huffman 編碼最大的差別就是，觀念上它能將一個符號以分數個位元的碼長作編碼。本論文在編碼的部分，也是基於算術編碼來實作的。

1.2 研究方法

為了比較壓縮效果的好壞，需要量測壓縮率以作為一個判斷標準，在此壓縮率的定義如下：

$$\text{壓縮率} = \frac{\text{壓縮後大小}}{\text{壓縮前大小}} \times 100\% \quad (1)$$

所以壓縮率愈小代表壓縮效果愈好。

由之前提到的文本資料壓縮方面的研究文獻，我們發現在剖析的部分，使用大字符集的觀念所得到的壓縮率通常都會比使用小字符集來得小。使用大字符集的研究，通常是從中文文章中，每次取兩個 byte 為 token，從英文文章每次取一個英文單詞(word)為 token。但是過去的研究，在取出一個英文詞彙後，只對該詞彙作零階的預測編碼，亦即只考慮各個詞彙在處理過文章中的出現機率，而沒有利用到詞彙之間的相關性。過去所以沒看到有人研究使用大字符集來作英文的高階預測編碼，其原因應是，所需要的記憶體非常大，且壓縮處理所花的時間也較長。因此我們便思考，若使用詞彙為 token 的大字符集來作英文的高階預測編碼，是否能達到更低的壓縮率？而它的相關問題是，使用龐大的資料結構的情況下，如何妥善管理記憶體、減少壓縮時間？也是需要去研究有效率的作法。

在剖析步驟，我們採用基於詞彙式(word-based)的方法，也就是以詞彙為 token，並將可能出現的詞彙分為三類：中文詞彙、英文詞彙、符號詞彙，因此可能從一篇文章中剖析出來的不同詞彙非常多，也就是這些可能詞彙所建立的詞符集，它的大小理論上是無限的，而實際上是有限但非常大。在模式步驟，本論文主要是使用最高階為二階的部分匹配預測模型觀念來實作，應用此模型來估算文章中所剖析出來的詞彙的機率，但由於詞符集的大小非常大，因此為了加速詞彙搜尋、詞彙機率計算、模型更新，我們用到樹(tree)及雜湊表(hash table)的資料結構來建構模型。在編碼步驟，本文採用最逼近熵值的算術編碼方法，來將目前詞彙依模型預測出的機率編成二進位碼。由於所找到的算術編碼軟體是針對小字符集(256 個字符)來設計的，因此本文修改算術編碼軟體，將計算上的數值範圍加大，以支援大字符集的算術編碼處理。

本文所製作的壓縮程式之流程，與一般文本資料壓縮的流程大致相同，如圖 1 所示，其中每個步驟的詳細處理方法，將在以下各節中說明。所製作的壓縮程式執行檔，可以從 http://guhy.csie.ntust.edu.tw/~adream/下載，提供測試之用。

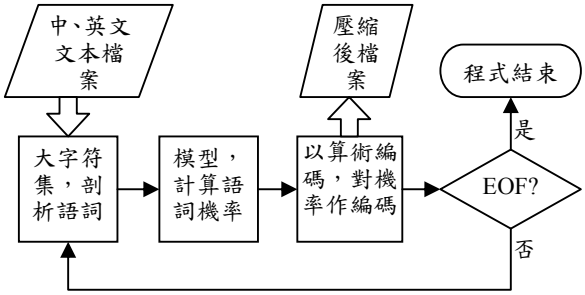

圖 1　詞彙式文本資料壓縮之流程

2. 大字符集之剖析、資料結構、算術編碼

2.1 基於大字符集之剖析

本研究以詞彙為 token，一次剖析出一個詞彙來作壓縮處理，我們將詞彙分成四類：中文詞彙、英文詞彙、常用符號詞彙、罕用符號詞彙。若取出的詞彙在詞符集(word alphabet)中沒出現過，則必須先對這個新詞彙的各個組成 byte 作編碼，編碼作法是，依該詞彙所屬類別按照規則將該詞的各組成 byte 以重新表示值(re-representation value)表示，而重新表示值的計算，則依據所有可能的 byte 值所成的字符集大小來訂定，以避免不必要的字符

集空間浪費。

中文單字剖析: 中文單字之剖析，我們依據 BIG5 碼的編碼規則，將連續兩個 byte 值取出來判斷是否為中文字，將符合表 1 中所列範圍的連續兩個 byte 剖析成一個中文單字。在此我們將全形的英文字元視為中文單字。

表 1　剖析中文單字詞彙之 byte 值範圍

	byte 值範圍		
第一個 byte	162	163~198, 201~248	249
第二個 byte	175~254	64~126, 161~254	64~126, 161~220

而每個中文單字的兩個重新表示值計算公式為：

$$R_1 = B_1 - k, \quad \begin{cases} k = 162, & \text{if } 162 \le B_1 \le 198 \\ k = 164, & \text{otherwise} \end{cases} \tag{2}$$

$$R_2 = B_2 - j, \quad \begin{cases} j = 64, & \text{if } 64 \le B_2 \le 126 \\ j = 98, & \text{otherwise} \end{cases} \tag{3}$$

公式(2)為第一個 byte B_1 的重新表示值 R_1 的計算公式，公式(3)為第二個 byte B_2 的重新表示值 R_2 的計算公式。

中文二字詞剖析: 本文將中文文章中常出現的中文二字詞剖析成一個中文詞彙，剖析的步驟如下：先假設連續出現的兩個中文單字詞是一個二字詞，而到詞符集中搜尋該二字詞是否已經存在詞符集中，若是存在，則一次將這兩個連續中文字剖析成一個中文二字詞，否則只取出第一個中文單字作為 token。我們並不是在作剖析時，附帶建立新的中文二字詞，至於將中文二字詞加入詞符集的作法，本文研究了兩種作法：靜態詞典方式，動態判斷方式。

靜態詞典方式是，在程式初始化時，直接將 20,638 個常用中文二字詞加入詞符集中，並將各個中文二字詞的出現次數設為 1。

動態判斷方式是，在將一個剖析出的詞彙編碼完畢時，檢查目前的詞彙 W_i 及前一個出現的詞彙 W_{i-1} 是否都是中文單字詞，若是的話，假設 W_{i-1} 的出現次數為 c_1，而(W_{i-1}, W_i)兩個單字詞一起出現的次數為 c_2，當條件($c_2 \ge 5$ 且 $c_2/c_1 \ge 0.5$)成立時，就將這兩個單字詞組合成一個中文二字詞，加入詞符集中，並將該二字詞的出現次數設為 c_2。

英文詞彙剖析: 英文詞彙的剖析是，一次判斷一個輸入的 byte，若該 byte 的 ASCII 值屬於表 2 中所列的範圍，則表示該 byte 是一個英文字元，我們將連續出現的英文字元剖析成一個英文詞彙，但最大長度不能大於 20，在此我們將阿拉伯數字也視為英文字元的一部分。

表 2　剖析英文字元之 byte 值範圍

字元	'0' ~ '9'	'A' ~ 'Z'	'a' ~ 'z'
Byte 值	48~57	65~90	97~122

各個英文字元的重新表示值計算公式如下：

$$R = B - i, \quad \begin{cases} i = 48, & \text{if } 48 \le B \le 57 \\ i = 55, & \text{if } 65 \le B \le 90 \\ i = 61, & \text{otherwise} \end{cases} \tag{4}$$

公式(4)將英文詞彙中每一個英文字元的 byte 值 B 轉換成該字元的重新表示值 R。

常用符號詞彙剖析: 常用符號詞彙是由連續的常用符號字元所組成，而總長度不能超過 20。我們將滿足表 3 中 byte 值範圍的字元及滿足表 4 中 byte 值組合的連續兩個 bytes 剖析成一個常用符號字元。

表 3　常用半形符號字元之 ASCII 範圍

字元	控制碼	標點符號
ASCII 碼	0~31	32~47, 58~64, 91~96, 123~126

表 4　常用全形符號字元之 byte 值範圍

	byte 值範圍
第一個 byte	161~162
第二個 byte	64~126, 161~174

各個常用符號字元的重新表示值計算公式如下：

$$R = B - i, \quad \begin{cases} i = 0, & \text{if } 0 \le B \le 47 \\ i = 10, & \text{if } 58 \le B \le 126 \\ i = 44, & \text{otherwise} \end{cases} \tag{5}$$

公式(5)將常用符號詞彙中每一個字元的 byte 值 B 轉換成該字元的重新表示值 R。

罕用符號詞彙剖析: 罕用符號詞彙就是由連續的罕用符號字元所組成，罕用符號詞彙的最大長度不能超過 4。表 5 列出罕用符號字元之 byte 值的範圍。

表 5　罕用符號字元之 byte 值範圍

	罕用符號字元
byte 值	127~255

各個罕用符號字元也以重新表示值表示，轉換公式如公式(6)所示。

$$R = B - 127 \tag{6}$$

每次剖析出一個新的詞彙，組成 bytes 經過重新表示後，會先編碼出一個詞彙層級的逃脫 token，接著再對組成字元作編碼，然後將此新詞彙插入詞符集中，詳細步驟之後會再介紹。

2.2　雜湊表(hash table)

在詞符集方面，我們使用雜湊表來加速搜尋及更新詞彙的計數資料，雜湊表主要的用途是可以將一個詞彙轉換成一個索引值(index)，若將一個詞彙的資訊存在這個索引值所指的位置上，之後若要搜尋某一個詞彙的資訊時，就不需作循序式(sequential)的搜尋，因而可加快搜尋速度。

雜湊函數(hash function): 雜湊函數的作用就是要將一個詞彙轉成一個索引值，雜湊函數的設定對雜湊表搜尋速度有很大的影響。在此我們定義兩個雜湊函數 f_1 及 f_2，公式如下：

$$f_1(w):\quad index_1 = (index_1 \times 613 + w_i)\,\%\,size \qquad i = 1, 2, \ldots, length \qquad index_1 = (index_1 + length)\,\%\,size \tag{7}$$

$$f_2(w):\quad index_2 = (index_2 \times 127 + w_i)\,\%\,size \qquad i = 1, 2, \ldots, length \qquad index_2 = (index_2 + 1)\,\%\,(size - 1) + 1 \tag{8}$$

其中 *size* 為雜湊表的大小，*w* 為所要轉換的字串，w_i 為字串中第 *i* 個字元，*length* 為 *w* 的長度。$index_1$ 與 $index_2$ 為經由兩個雜湊函數所求出的索引值，它們的初值設為 0，然後將字串中每一個字元 w_i 分別代入公式(7)及公式(8)的第一列後，得到介於 0 到 *size*-1 之間的索引值。之後，$index_1$ 再作公式(7)第 3 列的運算，$index_2$ 也多作公式(8)第 3 列的運算，以讓 $index_2$ 的範圍變為 1 至 *size*-1。本文利用 double hashing 的方法來求索引值。

double hashing: 當利用雜湊函數將目前所要加入的詞彙轉換成索引值時，若發現該索引值的位置上已經有另一個詞彙，這時就發生“衝突”(collision)的問題，必須再計算下一個索引值來存放目前這一個詞彙，解決衝突的方法有許多種，其中 double hashing 的效果不錯，因此我們採用這個方法來計算詞彙的索引值。

Double hashing 的方法需要用到兩個雜湊函數，我們利用之前定義的兩個雜湊函數來實作。公式如下：

$$new\_index_i = (index_1 + i \times index_2)\,\%\,size \tag{9}$$

其中 $index_1$ 為雜湊函數 f_1 所求出的索引值，$index_2$ 為雜湊函數 f_2 求出之索引值，new_index_i 表示以 double hashing 方法算出之第 *i* 次的索引值，*i* 的值從 1 至 *size*。若要保證公式(9)計算出來的索引值皆不同，即能夠對整個雜湊表的所有位置都訪問一次，其條件是，$index_2$ 要與 *size* 互質，因此我們將雜湊表的大小 *size* 取為質數。

負載係數(load factor): 負載係數的值界於 0 到 1 之間，當負載係數為 0.1，表示目前雜湊表中已經被用掉十分之一的空間，若負載係數超過一個門檻值時，就要從表中刪除一些資料或是增加雜湊表的大小以降低負載係數來加快搜尋，在此我們將負載係數之門檻值設為 0.72[17]。

2.3 各階字符集的結構

本文共使用 6 種字符集，如表 6 所示。每一個字符集代表某一類詞彙所使用到的字元集合。

表 6　六種字符集

種類	漢字第一 byte	漢字第二 byte	英文字元	常用符號字元	罕用符號字元	種類長度字元
字元個數	86	157	62	131	129	46
最大階數	1	1	0	4	4	3

表 6 中，“種類長度字元”字符集為所有可能的“種類”、“長度”字元重新表示值所成集合。由於一個剖析出的詞彙會屬於不同的種類及具有不同的長度，因此需要將一個新詞彙的種類及長度以一個“種類長度字元”表示，其重新表示值如表 7 所示。

表 7　種類長度字元之重新表示值

詞彙種類	中文	英文	常用符號	罕用符號
詞彙長度	1~2	1~20	1~20	1~4
重新表示值	0~1	2~21	22~41	42~45

我們對於上述 6 種字符集裡的字元，希望所有字元都能在 0 階模型中求得機率，因此在 0 階模型中將這 6 種字符集的所有字元皆初始化為出現 1 次。為了增加搜尋速度，我們在 0 階模型中，將每種字符集的樹建成一個平衡樹(balanced tree)，而高階模型字符集是從 0 階模型字符集中每個樹節點上的指標指出，其結構如圖 2 所示。

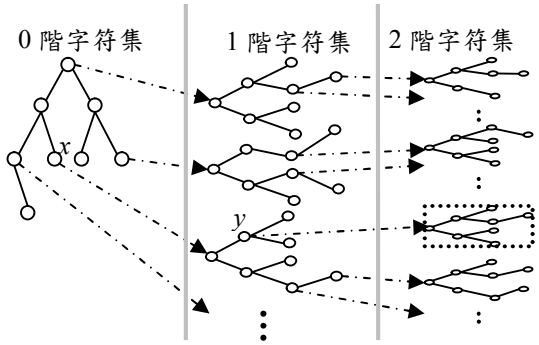

圖 2　各階預測模型的字符集結構

圖 2 中，假設前兩個出現字元為 *x*、*y*，若要計算目前字元的 2 階模型機率，則必須先取得在 *x*、*y* 出現情形下的 2 階搜尋樹，取得的步驟為：先從 0 階字符集的樹中找出 *x* 節點，接著由 *x* 節點中指向 1 階的指標得到在 *x* 出現情形下的 1 階搜尋樹，並且在此 1 階的樹中找出 *y* 節點，便可由此 *y* 節點中指向 2 階的指標，得到我們所要找的搜尋樹，即圖 2 中右邊虛線方框部分。

2.4 各階詞符集的結構

本文將所有剖析出的不同詞彙儲存在 0 階模型詞符集中，用以計算各個詞彙的 0 階預測機率。當要把詞彙加入 0 階模型詞符集時，先以雜湊函數算出索引值，再將此索引值插入詞符集的二元搜尋樹中，並將該詞彙資訊存在索引值所指的雜湊表格子(entry)上，此外每個雜湊表格子也儲存一個指向高階模型詞符集之指標及該高階詞符集的一些資訊(如：“出現詞彙個數”、“詞符集大小”、“出現一次的詞彙個數”等)，如圖 3 所示的，就是各階模型的詞符集結構。

圖 3 中，假設前兩個出現的詞彙的索引值為 *x*、*y*，為了要取得在 *x*、*y* 出現情形下的 2 階模型搜尋樹，首先我們由雜湊表中 *x* 位置指向 1 階的樹，接著在此 1 階的搜尋樹中以二元搜尋的方式

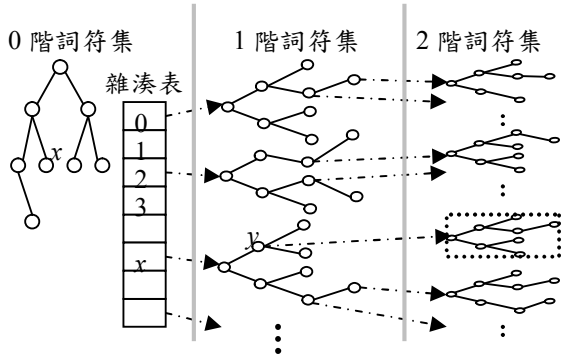

圖 3　高階詞符集之結構

找出 y 節點，便可由此 y 節點得到指向 2 階搜尋樹的指標，如圖 3 中右邊虛線方框部分。

最近最少使用串列(LRU list): 為了管理各階詞符集所佔用的空間，我們使用"最近最少使用串列"(LRU list)資料結構。一開始先將樹節點的空間以 LRU 串列串在一起，當需要用到樹節點時，便由 LRU 串列的開頭取空間來使用，而刪除某個樹節點時，也放回 LRU 串列的開頭部分，而在每次更新完某個樹節點後，則將該樹節點放到 LRU 串列的尾端。如此一來，便能統一管理所有二元搜尋樹的樹節點，且樹節點所佔的空間不需持續增加。

在本研究中，我們用到兩個 LRU 串列：LRU_1 及 LRU_2。其中 LRU_1 將所有 0 階的樹節點全部串在一起管理，而 LRU_2 將所有 1 階、2 階的樹節點串在一起管理。在此我們設定 LRU_1 長度為 87,719 個節點，因為雜湊表的大小初始化為 87,719，當雜湊表的負載係數超過 0.72 時，我們將那些在串列前半段的節點刪除，也就是將 0 階詞符集中最近較不常用的詞彙從表中刪除，而且我們要到高階的詞符集中檢查是否有這些被刪除的詞彙，有的話也必須跟著刪除。至於 LRU_2 的長度，在此設為 1,200,000 個節點，即 1 階、2 階中所有樹節點的個數總和為 1,200,000，當空間不夠時就刪除 LRU_2 串列中較不常用的樹節點，在此我們一次就刪除 600,000 個樹節點。

2.5　算術編碼

算術編碼主要的優點是觀念上可將每個符號用分數個 bits 來表示，不像 Huffman 編碼，其每個符號必須用整數個 bits 來編碼。關於算術編碼詳細的步驟及實作方法可參考文獻[2, 6, 18]。

支援大字符集之算術編碼: 本文在算術編碼的部分，是採用 M. Nelson[18]提供之算術編碼程式碼來作改進。Nelson 將算術編碼以整數區間的表示方式來實作，也就是將符號的機率以符號的區間下限、區間上限及所有符號的機率區間表示，這些區間值都必須是整數。在解碼時只要有所有符號的機率區間之資訊，便可由編碼後的資料計算出一個值，只要求出該值是落在哪一個符號的區間，便可解出原始符號。

在 Nelson 的原始算術編碼程式中，是以 16 位元的整數來表示數值，即初始整數區間為 0 至 65,535，若所有符號的區間總和超過 65,535 的四分之一時，算術編碼的過程就會會出問題，因此我們將其改為 32 位元的整數來表示及作運算，如此詞符集中各詞彙的出現次數總和之數值才可被正確表示與處理。

為了求得一個符號的整數值累積區間，及所有符號的總和區間範圍，在 Nelson 的程式中[18]，直接以線性累加的方式來求，也以線性更新的方式來更新所有符號的累積次數，因為其字符集大小只有 256，所以最多每次只要作 256 次的累加及更新處理。但是若處理的是大字符集，用線性累加的方式會非常耗費時間。因此，在論文[8]中提到的針對大字符集求符號的累積次數及更新符號累積次數的方法，是以 Fenwick 在 1994 年提出之資料結構來實作[19]，此外教科書[21]裡也介紹了一種作法，它們對每個符號所花的時間為 $\Theta(\log N)$，N 為字符集的大小。因此本文採取以"支援累積之二元搜尋樹"來求各符號的整數值累積區間。

支援累積之二元搜尋樹: 二元搜尋樹的好處就是能夠快速的在樹中找出某個符號，而支援累積之二元搜尋樹(accumulation-supported binary search tree)不但能快速的找出樹中某個符號，並能在找到該符號的同時算出該符號之累積次數，其方法是在每個樹節點中加入一個欄位，用來存放該節點之左子樹所有符號出現次數之總和。

多了這個欄位的資訊，我們便可以快速的求出某個符號的出現次數累積區間、模型中所有符號的出現次數總和、及某個累積次數值在模型中所對應的符號。為了維持此欄位數值的正確性，我們必須在插入樹節點、刪除節點、及所有節點出現次數減半時，檢查、更新這個欄位的值。

3. 機率預測模型

3.1 基於部分匹配之預測模型

基於部分匹配之預測模型(簡稱 PPM)的觀念是由 Cleary 及 Witten 提出的[11]，本文基於他們的觀念來研究詞符集中的符號出現機率的估計方法。PPM 模型主要是利用之前若干個符號來預測目前符號的出現機率，由於有時會出現零頻的情況，因此我們也必須考慮、解決零頻的問題。

模型的階數: 模型的階數(order)就是往前參考的符號個數，若某個模型是以前 1 個符號來預測目前符號，則稱它為 1 階模型，若一個模型不參考之前符號，只用目前符號在壓縮過的文章裡的出現次數來估計機率，則稱它為 0 階模型。當模型的階數愈高時，所需建立的搜尋樹個數就會愈多，例如符號的個數為 n，則 0 階模型只需要一個搜尋樹，1 階模型最多需要 n 個搜尋樹，2 階模型最多需要 n^2 個搜尋樹，以此類推。PPM 模型實際上是把多個

不同階數的模型結合起來，因此 PPM 模型的的“最大階數”指的是，最高階模型往前參考的符號個數。

本文在各種字符集情況分別設定的模型最大階數值如表 6 所示，設定方式是以實驗結果來決定，對於大部分的文本資料，皆適合使用如表 6 裡的最大階數設定。其中，英文字元若不使用之前出現的英文字元來作預測，反而會得到較低的壓縮率；常用符號之間的關聯性較英文高許多，因此我們設定常用符號字元往前考慮最多 4 個常用符號字元來預測目前的；種類長度字元並不往前參考種類長度字元，因為長度與長度之間的關係較小，而是種類與種類之間的關係較大，因此種類長度字元只往前參考種類的部分，最多往前參考 3 個種類；中文第一個字元是參考前 1 個中文第一字元來預測，因為中文常用字的第一個字元通常都集中在某幾個字元，因此不同中文字的第一個字元之間關聯性較大；中文第二個字元與前一個中文的第二個字元，關聯性非常低，因此我們用同一個中文字的第一個字元來預測第二個字元。

逃脫符號: 在 0 階字元模型中，程式初始化時便將所有可能的符號都加入了，並且都設定出現次數為 1，因此各個符號都可以從 0 階字元模型得到一個機率。但是在高階字元模型中，由於字符集搜尋樹初始時為一個空樹，只有在遇到符號時才會被加入搜尋樹，因此對於一個還未被加入高階字元模型搜尋樹中的符號，便無法求出它的高階模型機率，這種情形在詞彙模型裡更是經常遇到。當遇到這種情形時， PPM 模型中的解決方法是，在每個高階模型字符集內加入一個稱為“逃脫符號”(escape symbol)之特殊符號，如此當遇到要估計機率的符號不在搜尋樹中，便換成對“逃脫符號” 作編碼，藉以告知解碼端有一符號不存在於搜尋樹中，並且要降到下一階的模型來處理。

字符集逃脫機率: 逃脫符號的出現機率的訂定，必須根據詞(或字)符集中所有符號的出現情形隨時改變，才能盡量減少逃脫符號對壓縮率造成的影響。當面對的是字符集時，我們參考算術編碼程式中用來計算逃脫符號機率的方法[18]，公式如下：

$$\text{逃脫符號出現次數} = \frac{(\text{字符集大小} - \text{出現符號個數}) \times \text{出現次數總和}}{\text{字符集大小} \times \text{最大出現次數}} \quad (10)$$

公式(10)中考慮到 2 個因素：(a)出現的符號個數愈多，逃脫符號的出現次數就會愈小，若所有符號都出現過，就會使逃脫次數變成 0。(b)當“出現符號個數”很小，且(出現次數總和/最大出現次數)的值也很小時，表示出現的符號個數很少，但某個符號出現非常多次，此時該符號的出現機率會非常大，因此逃脫次數要相對減少。若最後算出的逃脫次數是介於 0 到 1 之間的小數，則必須將逃脫次數設為 1。只要模型中有符號尚未出現過，就一定要保留一些出現次數給逃脫符號，否則無法處理符號不在搜尋樹中的情況。

詞符集逃脫機率: 詞符集與字符集的特性不同，無法用公式(10)來計算，其中最大的不同點，就是詞彙符號的數量事先並不知道，而一直可能有新的詞彙會出現，可能的詞彙個數理論上是無限的。在此我們參考一些論文所使用的方法來計算逃脫機率。

第一個考慮的方法為 Moffat 提出之 Method C [20]，他將之前的方法 Method A 及 Method B 作改進。此外，他也針對英文之詞彙式 PPM 模型作實驗，發現 Method C 不管在小字符集或大字符集中，都能得到比 Method A 及 Method B 還要低的壓縮率，Method C 的公式如下：

$$\text{逃脫機率} = \frac{\text{出現符號個數}}{\text{符號出現次數總和} + 1} \quad (11)$$

第二個考慮的方法為 Witten 及 Bell 提出之 Method X[7]。在他的論文中提到，Method X 幾乎對大字符集或小字符集皆有不錯的壓縮率。我們參考他們的公式並遵循建議，以避免分子等於分母及分子分母皆為 0 的情形，公式如下：

$$\text{逃脫機率} = \frac{\text{出現一次的符號個數} + 1}{\text{符號出現次數總和} + 2} \quad (12)$$

第三個考慮的方法為 Moffat、Neal、Witten 提出之針對大字符集結構之逃脫機率計算方法[8]，在文獻中稱為 Method AX，即對論文[7]中所提的 Method X 作一些修改，其公式如下：

$$\text{逃脫機率} = \frac{\text{出現一次符號數} + 1}{\text{符號出現次數總和} + \text{出現一次符號數} + 1} \quad (13)$$

使用這三個公式之一算出逃脫機率後，再依公式(14)轉換為逃脫次數，公式如下：

$$\text{逃脫次數} = \text{符號出現次數總和} \times \frac{\text{逃脫機率}}{1 - \text{逃脫機率}} \quad (14)$$

兩段式 PPM 模型: PPM 觀念的實作，在此分為兩個階段：詞彙階段及字元階段。其中詞彙階段負責計算詞彙機率，而字元階段則計算一個新的詞彙中各個字元的機率，而整個處理流程如圖 4 所示。

在圖 4 裡，一個被剖析出的詞彙，我們首先以 2 階的詞彙模型來預測機率，如果 2 階模型裡存在搜尋樹，並且目前要編碼的詞彙在搜尋樹中，則求出目前詞彙在 2 階詞彙模型中的機率，接著以算術編碼依此機率作編碼，最後更新階數高於目前階數的所有高階模型，然後繼續對下一個剖析出來的詞彙作處理；如果目前要編碼的詞彙不在搜尋樹中，則求出此詞符集中的逃脫符號的機率，接著以算術編碼編此機率，並將模型階數下降一階。

當遇到一個新詞彙時，會在 0 階模型的搜尋樹中找不到，因此階數會降至-1 階，此時我們先將新詞彙加到 0、1、2 各階的詞符集中，接著切換成字元階段，將組成新詞彙的所有字元一一編碼。首先對此新詞彙的種類及長度編碼，以告知解碼端接下來所要參考的字符集種類及新詞彙中字元的個數，接下來才編詞彙的所有字元。編每一個字元的方式則與編每一個詞彙的方式類似，不同的是，

我們在 0 階字符集已加入所有的字元，因此不會有降到-1 階的情形，所有符號至少都會在 0 階得到其機率值。當所有字元都編完時，便回到剖析的步驟去剖析出下一個詞彙來編碼。

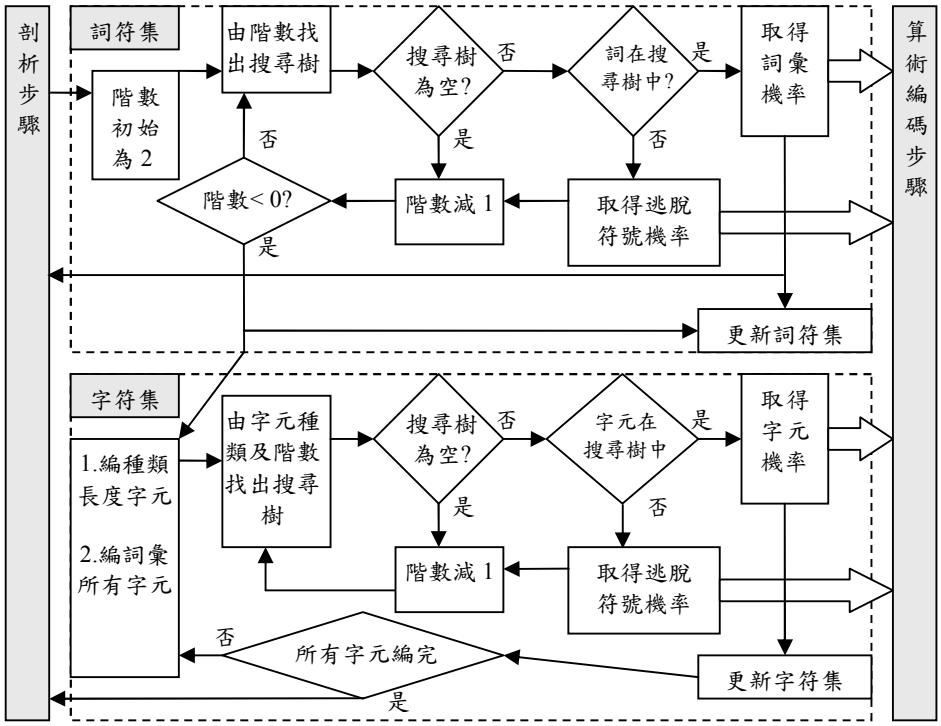

圖 4 兩段式 PPM 模型的主要處理流程

3.2 混合式模型

混合式模型與 PPM 模型之間的差異是，混合式模型把數個不同階數的模型，以不同權重混合在一起，用混合後的模型計算目前符號的機率，並且編碼完後會同時更新各階的模型。我們僅在詞彙階段套用混合式模型，其資料結構大致相同，同樣有 0、1、2 階模型的詞符集搜尋樹。不過，在字元階段的編碼，則和前一子節(3.1 節)介紹的完全相同，混合式模型的處理流程如圖 5 所示。

我們將三個模型依權重混合後，必須確保混合後之模型中每個符號所得到的累積次數區間皆各自獨立，不會互相重疊，解碼時才不會發生解碼錯誤的情形。使用混合式模型的相關研究在論文[10, 21]中皆有提到。

模型範例: 在此我們舉一個混合式模型的範例，來介紹混合的作法及驗證各符號的累積出現次數區間不會重疊。為了說明之方便，在此只舉兩個模型作混合的例子，並將兩個模型之權重都設為 1，關於三個模型及不同權重之混合方式，則可由此例子引伸出去。

假設兩個模型中各符號之出現次數如表 8 所示，模型一中有 5 個符號，模型二中有 4 個符號，有些符號只出現在其中一個模型，有些則兩個模型都出現過。兩個模型之混合就是將兩個模型分別求出之累積區間加在一起，

表 8 兩模型內各符號之出現次數

模型一		模型二	
符號	出現次數	符號	出現次數
s_1	3	s_3	2
s_2	4	s_4	1
s_4	3	s_5	3
s_6	5	s_6	6
s_7	2		

混合的計算過程如表 9 所示，首先分別得到各個模型中的符號的累積次數區間，若符號不在模型中，其累積次數區間必須參考前一個符號的區間右邊界值。例如表 9 中模型一中的符號 s_3，其出現次數為 0，由於 s_2 的累積區間右邊界值為 7，因此 s_3 之累積區間為 7 ~ 7，這樣便能保證混合模型中每個符號的累積出次數皆不會重疊，如表 9 的第三欄所示。

表 9 混合式模型之累積出現次數計算

模型一		模型二		混合模型	
符號	累積出現次數	符號	累積出現次數	符號	累積出現次數
s_1	0 ~ 3	s_1	0 ~ 0	s_1	0 ~ 3
s_2	3 ~ 7	s_2	0 ~ 0	s_2	3 ~ 7
s_3	7 ~ 7	s_3	0 ~ 2	s_3	7 ~ 9
s_4	7 ~ 10	s_4	2 ~ 3	s_4	9 ~ 13
s_5	10 ~ 10	s_5	3 ~ 6	s_5	13 ~ 16
s_6	10 ~ 15	s_6	6 ~ 12	s_6	16 ~ 27
s_7	15 ~ 17	s_7	12 ~ 12	s_7	27 ~ 29

特性證明: 我們欲證明混合模型中的所有符號之累積出現次數區間皆不重疊，假設在模型一中，第 i 個符號 s_i 的累積出現次數為 $xlow_i \sim xhigh_i$，在模

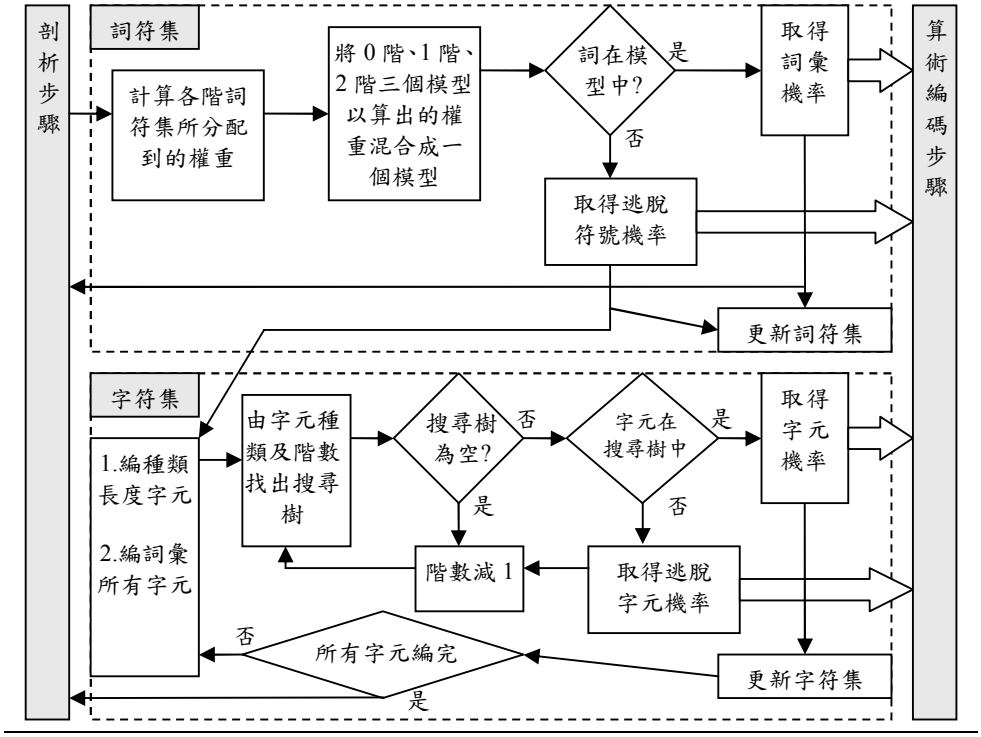

圖 5 混合式模型的主要處理流程

型二中，s_i 的累積出現次數為 $ylow_i \sim yhigh_i$。在混合模型中，令 s_i 的累積出現次數為 $zlow_i \sim zhigh_i$，其中 $zlow_i = xlow_i + ylow_i$，$zhigh_i = xhigh_i + yhigh_i$。

因為模型一中所有的符號的累積出現次數區間皆不重疊，因此對所有模型一中的符號 s_i，皆滿足$(xlow_i \geq xhigh_{i-1})$且$(xhigh_i \leq xlow_{i+1})$條件。模型二中所有符號的累積出現次數區間也不重疊，則對所有模型二中的符號 s_i，滿足$(ylow_i \geq yhigh_{i-1})$且$(yhigh_i \leq ylow_{i+1})$之條件。

由以上兩個條件相加得到$(xlow_i + ylow_i \geq xhigh_{i-1} + yhigh_{i-1})$且$(xhigh_i + yhigh_i \leq xlow_{i+1} + ylow_{i+1})$，即$(zlow_i \geq zhigh_{i-1})$且$(zhigh_i \leq zlow_{i+1})$。由此可知混合模型中所有符號的累積出現次數區間皆不重疊。

<u>權重計算</u>: 由於各個模型的重要性不同，因此要賦予各個模型不同的權重值。作混合時，可先將各個模型之符號出現次數乘上各自的權重，再將累積出現次數區間相加。如此，混合後模型的機率分佈，將會與權重值最大的原始模型最相似，但也保留了一些權重低的原始模型中機率分佈之特性。

所謂各個模型的權重，是假設各個模型的所有符號的出現次數總和皆相同的情形下設定的。因此在確定模型間的權重關係時，必須同時考慮到各個模型中的所有符號的總和出現次數。在本文中，我們將 2、1、0 階三個詞彙模型作混合，假設三個模型中各自的符號出現次數總和依序為 $total_2$、$total_1$、$total_0$，而依據模型的重要性所算出各模型的權重值假設為 w_2、w_1、w_0，接著我們以 0 階模型作為基準，令其權重值為 1，則改變後的模型權重值依序為 $w_2' = w_2 / w_0$，$w_1' = w_1 / w_0$，$w_0' = 1$，再考慮各模型內的符號總和出現次數的因素之後，最後得到各階模型實際的權重公式如下：

$$\begin{aligned} weight_2 &= w_2' \times \frac{total_0}{total_2} = \frac{w_2}{w_0} \times \frac{total_0}{total_2} \\ weight_1 &= w_1' \times \frac{total_0}{total_1} = \frac{w_1}{w_0} \times \frac{total_0}{total_1} \\ weight_0 &= 1 \end{aligned} \tag{14}$$

經由實驗測試，我們發現若$(w_2'+w_1')$之值太大，對壓縮率反而會造成不好的影響，因此我們對此值作限制，當$(w_2'+w_1') >$ 最大值 max_w 時，令 $w_2'' = max\_w \times w_2' / (w_2'+w_1')$，而 $w_1'' = max\_w \times w_1' / (w_2'+w_1')$，設定此最大值的另一個原因是，用以預防混合後模型所算出的符號總和出現次數會超過算術編碼所能接受的範圍，因為混合後模型之符號總和出現次數，$(1 + w_2' + w_1') \times total_0$，可保證小於$(1 + max_w) \times total_0$。此外每次在更新完 0 階之模型後，可據以檢查$(1 + max_w) \times total_0$是否超過算術編碼之範圍，若超過，則對零階模型裡的符號，作出現次數減半之處理。

關於 w_2、w_1、w_0 權重值的設定，我們參考論文[10, 21]裡使用之計算方法，該方法是依據逃脫機率來計算各階模型的權重，假設第 i 階模型之逃脫機率為 esc_i，則權重計算公式如下：

$$\begin{aligned} w_2 &= 1 - esc_2 \\ w_1 &= esc_2 \times (1 - esc_1) \\ w_0 &= esc_2 \times esc_1 \times (1 - esc_0) \end{aligned} \tag{15}$$

將式(15)代入式(14)後，得到如下之權重計算公式：

$$weight_2 = \frac{1 - esc_2}{esc_2 \times esc_1 \times (1 - esc_0)} \times \frac{total_0}{total_2}$$

$$weight_1 = \frac{esc_2 \times (1-esc_1)}{esc_2 \times esc_1 \times (1-esc_0)} \times \frac{total_0}{total_1} \quad (16)$$

$$weight_0 = 1$$

接著考慮一個情況，如果剖析出的詞彙是混合後模型裡不存在的一個新的符號，此時要對逃脫符號編號，以告知解碼端目前的符號不存在於模型中，混合後模型的逃脫符號的機率估計值是 $esc_2 \times esc_1 \times esc_0$，其對應的出現次數公式為：

$$逃脫次數 = \frac{esc_2 \times esc_1 \times esc_0}{esc_2 \times esc_1 \times (1-esc_0)} \times total_0 \quad (17)$$

由式(16)、式(17)可知，若使用的逃脫機率估計方法不同，得到的權重值及逃脫符號的出現次數也會不同。

4. 測試實驗與結果

本文已將先前提到的各種資料結構、預測模型實作成為程式，然後進行各項實驗。我們選擇用來作測試的文本資料分別為 7 個中文檔案、5 個英文檔案，每個檔案的內容及檔案大小如表 10、表 11 所示。

表 10　中文測試檔內容說明

檔案名稱	檔案大小 (bytes)	內容
達文西	632,685	翻譯小說－達文西密碼
絕代雙嬌	1,679,292	古龍武俠小說
笑傲江湖	2,018,872	金庸武俠小說
雜誌	3,198,900	華夏文摘 2003 年雜誌內容
現代小說	3,994,835	哈利波特三集、瓊瑤三篇小說、射雕英雄傳
古典小說	4,557,249	紅樓夢、三國演義、水滸傳
新聞	6,163,300	中央社新聞，1995 年 9 至 11 月

表 11　英文測試檔內容

檔案名稱	檔案大小 (bytes)	內容
Women	1,039,390	Little Women
Anne	1,969,393	Anne of Avonlea, Anne of Green Gables, 及 Anne of the Island
Ring	2,865,136	The Lord of the Rings, 四集
Bible	4,477,959	聖經
Harry	5,589,522	Harry Potter, 四集

實驗時所使用的測試平台為 P4 1.3G 的個人電腦，關於壓縮效果的好壞，我們使用公式(1)的壓縮率計算方式來評估。當本文實作的程式執行時，最大使用之記憶體量為 71.8 Mbytes，其中(a)初始的 6 個 0 階字符集與一個 0 階詞符集所用的樹節點共有 87,719 個，佔 4.4 Mbytes；(b) 1 階、2 階詞符集搜尋樹共有 1,200,000 個樹節點被初始化，佔 55.2 Mbytes；(c)壓縮時產生的高階字符集及高階詞符集等，佔 12.3 Mbytes。

4.1 基於部分匹配模型之實驗

實驗裡各字符集的最大階數設定如表 6 所示，另外 0 階詞符集大小為 87,719，1、2 階詞符集裡可用的樹節點總和為 1,200,000，雜湊表的最大負載係數設為 0.72。

模型最大階數實驗: 詞符集的階數愈大，代表往前參考的詞彙愈多，對詞彙的預測愈準確。經由此實驗中，我們可知道模型的最大階數對壓縮率所造成的影響，實驗裡我們將模型的最大階數由 0 至 2 增加，所有的逃脫機率計算方法統一使用 Method C，即公式(11)，實驗結果如表 12 所示。

表 12　模型最大階數對壓縮率的影響

檔案	原始大小	0 階	1 階	2 階
達文西	632,685	320,572	232,393	**225,243**
絕代雙嬌	1,679,292	851,772	601,766	**562,951**
笑傲江湖	2,018,872	1,085,100	810,809	**761,033**
雜誌	3,198,900	1,751,761	1,321,057	**1,254,143**
現代小說	3,994,835	2,089,575	1,557,076	**1,475,107**
古典小說	4,557,249	2,590,775	2,005,401	**1,897,730**
新聞	6,163,300	3,263,283	2,012,526	**1,698,309**
中文總計	22,245,135	11,952,838 (53.7%)	8,541,028 (38.4%)	**7,874,516 (35.4%)**
Women	1,039,390	349,821	295,428	**279,334**
Anne	1,969,393	658,566	545,810	**501,070**
Ring	2,865,136	922,357	750,779	**681,617**
Bible	4,477,959	1,393,798	1,072,297	**870,833**
Harry	5,589,522	1,793,672	1,461,119	**1,306,207**
英文總計	15,941,400	5,118,214 (32.1%)	4,125,433 (25.9%)	**3,639,061 (22.8%)**

由表 12 之實驗結果可知，若使用較高階的模型來預測機率，可得到較低的壓縮率，不過隨著階數增加，其改善程度會減少。由 0 階增加至 1 階時，中文檔案壓縮率降低了 15.3%，英文檔案則降低了 6.2%；而由 1 階增加至 2 階時，中文檔案壓縮率降低了 3.0%，英文檔案降低 3.1%。因此我們將模型最大階數設定為 2 階。

逃脫機率估計方法實驗: 字符集的逃脫符號機率估計方法如公式(10)所示，而詞符集的逃脫機率，我們實驗三種估計方法，分別為公式(11)、(12) 與(13)，在此分別稱為 method C、method X、method AX。這三種逃脫機率估計方法，分別得到的壓縮率如表 13 所示。

表 13　逃脫機率估計方法對壓縮率的影響

檔案	原始大小	Method C	Method X	Method AX
達文西	632,685	225,243	**223,938**	227,291
絕代雙嬌	1,679,292	562,951	**559,559**	564,362
笑傲江湖	2,018,872	761,033	**756,662**	764,872
雜誌	3,198,900	1,254,143	**1,247,224**	1,264,507
現代小說	3,994,835	1,475,107	**1,470,709**	1,486,367
古典小說	4,557,249	**1,897,730**	1,898,511	1,922,270
新聞	6,163,300	1,698,309	**1,688,695**	1,701,629
中文總計	22,245,135	7,874,516 (35.40%)	**7,845,298 (35.27%)**	7,931,298 (35.65%)
Women	1,039,390	279,334	**278,237**	278,370
Anne	1,969,393	501,070	**499,258**	499,406

Ring	2,865,136	681,617	**678,655**	678,751
Bible	4,477,959	870,833	**869,747**	869,860
Harry	5,589,522	1,306,207	**1,302,091**	1,302,309
英文總計	15,941,400	3,639,061 (22.83%)	**3,627,988 (22.75%)**	3,628,696 (22.76%)

由表 13 可以看出，除了"古典小說"以外，所有的測試檔，利用 method X 方法來估計詞符集之逃脫符號的機率，都可以得到最低的壓縮率。對於中文檔案來說，method X 的壓縮率比 method C 好 0.13%；對英文檔案來說，method X 僅比 method AX 好 0.01%。

4.2 混合式模型之實驗

權重決定方法實驗： 我們設定各階模型的權重的計算公式如公式(16)所示，而混合後的模型之逃脫符號出現次數計算如公式(17)所示，並利用三種逃脫機率估計的公式，即公式(11)、(12)、與(13)，來求各階模型混合時的權重，在此我們分別稱為 method C 權重、method X 權重、method AX 權重。另外在公式(14)中提到 1 階與 2 階的權重總和的最大值，目前先將它設為 512，如此得到的實驗結果如表 14 所示。

表 14　模型權重計算方法對壓縮率的影響

檔案	原始大小	Method C	Method X	Method AX
達文西	632,685	**220,734**	223,332	229,136
絕代雙嬌	1,679,292	**551,781**	560,365	570,435
笑傲江湖	2,018,872	**744,280**	755,327	771,829
雜誌	3,198,900	**1,224,731**	1,240,359	1,272,701
現代小說	3,994,835	**1,455,538**	1,479,446	1,509,807
古典小說	4,557,249	**1,874,956**	1,905,530	1,946,705
新聞	6,163,300	**1,687,087**	1,715,170	1,737,650
中文總計	22,245,135	**7,759,107 (34.88%)**	7,879,529 (35.42%)	8,038,263 (36.13%)
Women	1,039,390	**277,180**	279,907	283,574
Anne	1,969,393	**499,745**	504,647	510,356
Ring	2,865,136	**680,763**	687,074	694,039
Bible	4,477,959	**876,786**	887,165	891,255
Harry	5,589,522	**1,307,935**	1,319,825	1,332,339
英文總計	15,941,400	**3,642,409 (22.85%)**	3,678,618 (23.08%)	3,711,563 (23.28%)

由實驗結果可看出，對所有的測試檔，使用 method C 計算出的權重可以得到最好的壓縮率，對中文檔案來說，method C 比 method X 好 0.54%；對英文檔案來說，method C 比 method X 好 0.23%。因此我們採用 method C 來計算混合式模型中各階模型的權重。

高階模型權重最大值實驗： 之前曾經提到，我們必須對公式(14)中第 1、2 階模型的權重總和作限制，以預防混合後模型的符號出現次數總和會超過算術編碼所能接受的範圍。在此我們以 method C 來計算權重，探討減少此值對壓縮率之影響，實驗結果如表 15 所示。

表 15　最大總和權重對壓縮率的影響

檔案	原始大小	16	32	64
達文西	632,685	220,878	220,823	220,803
絕代雙嬌	1,679,292	**551,520**	551,568	551,648
笑傲江湖	2,018,872	**743,990**	744,136	744,247
雜誌	3,198,900	1,225,016	1,224,959	1,224,878
現代小說	3,994,835	**1,453,274**	1,454,689	1,455,344
古典小說	4,557,249	**1,871,247**	1,873,546	1,874,644
新聞	6,163,300	1,689,706	1,687,934	1,687,644
中文總計	22,245,135	**7,755,631 (34.86%)**	7,757,655 (34.87%)	7,758,930 (34.88%)
Women	1,039,390	277,091	**276,876**	276,902
Anne	1,969,393	499,772	**499,203**	499,211
Ring	2,865,136	681,419	680,244	**680,092**
Bible	4,477,959	881,720	878,372	877,127
Harry	5,589,522	1,309,273	1,307,066	**1,306,789**
英文總計	15,941,400	3,649,275 (22.89%)	3,641,761 (22.84%)	**3,640,121 (22.83%)**

由表 15 可以知道，對大部分的測試檔來說，當把權重和的最大值設定為一個較小的值，並不會使壓縮率上升太多，而且權重和最大值愈小，0 階模型所能接受的出現次數總和值就可以愈大。對中文檔案來說，在權重和值最大為 16 或更小時，能得到較好的壓縮率，但效果並不顯著；對英文檔案來說，設權重和最大值為 64 時，壓縮率較好，因此我們設定權重和最大值為 32，如此對中、英文檔案皆能得到不錯的效果，且 0 階詞符集所能接受的出現次數總和最大值為 1,073,741,823 / 32，即 33,554,431，足夠用以處理大字符集的情形。

4.3 不同詞典之實驗

中文檔案實驗： 在 2.1 節中，我們提到剖析中文詞彙時，一次可將兩個中文字剖析為一個 token，而將中文二字詞加入 0 階詞符集中的方法可分為動態判斷及靜態詞典兩種。我們使用此兩種方法來將中文二字詞加到詞符集中，使用混合式模型來作壓縮率實驗，實驗結果如表 16 所示。

表 16　詞符集加入中文二字詞之方式

檔案	未處理	動態判斷	靜態詞典	動態+靜態
達文西	**220,823**	221,650	225,210	226,035
絕代雙嬌	**551,568**	552,139	555,324	556,303
笑傲江湖	**744,136**	745,613	749,088	750,517
雜誌	1,224,959	1,223,279	**1,222,297**	1,223,062
現代小說	**1,454,689**	1,454,691	1,458,434	1,460,258
古典小說	**1,873,546**	1,874,830	1,878,154	1,879,768
新聞	1,687,934	1,677,023	1,663,975	**1,660,851**
中文總計	7,757,655 (34.87%)	**7,749,225 (34.84%)**	7,752,482 (34.85%)	7,756,794 (34.87%)

由表 16 中壓縮率數值顯示，中文文章中，只有"雜誌"及"新聞"兩個檔案在加入中文二字詞後壓縮率有改善，但改善不大，其他的檔案則是不加入中文二字詞於詞符集中壓縮率較好。

英文檔案實驗： 對於英文檔案的壓縮，我們在程式

初始化時，將靜態詞典的 20,297 個常用的英文詞彙加入 0 階的詞符集中，並且將它們的出現次數設為 1。在此分別探討其對部分匹配預測模型、混合式模型的影響，實驗結果如表 17、18 所示。

表 17　PPM 模型加入靜態英文詞彙之壓縮率

檔案	原始大小	未處理	靜態式詞典
Women	1,039,390	278,237	**275,333**
Anne	1,969,393	499,258	**495,122**
Ring	2,865,136	678,655	**676,667**
Bible	4,477,959	869,747	**870,682**
Harry	5,589,522	1,302,091	**1,296,888**
英文總計	15,941,400	3,627,988 (22.75%)	**3,614,692 (22.67%)**

表 18　混合模型加入靜態英文詞彙之壓縮率

檔案	原始大小	未處理	靜態詞典
Women	1,039,390	276,876	**268,950**
Anne	1,969,393	499,203	**488,472**
Ring	2,865,136	680,244	**670,499**
Bible	4,477,959	878,372	**869,413**
Harry	5,589,522	1,307,066	**1,292,107**
英文總計	15,941,400	3,641,761 (22.84%)	**3,589,441 (22.52%)**

由表 17、18 之實驗結果可知，在 0 階詞符集中加入靜態英文詞彙，對於兩種預測模型都能減少壓縮率。未加入靜態英文詞時，部分匹配預測模型的壓縮率比混合式模型好 0.09%，但加入靜態英文詞後，混合式模型反而比部分匹配預測模型的壓縮率好 0.15%；由此可知，0 階詞符集中詞彙個數較多時，使用混合式模型能得到較好的壓縮率。

4.4　不同壓縮程式之比較

由之前的實驗結果得知，本文研究的所有方法中，使用混合式模型，對中文及英文檔案皆能得到不錯的壓縮率。因此我們拿混合式模型所實作之程式來和他人的壓縮程式作比較，這裡所謂的他人程式包括 GZIP[22]、bzip2[23]、及 PPMd[12]等壓縮程式，其中 GZIP 是應用 LZ77 的技術所作的壓縮程式；bzip2 是基於 Burrows-Wheeler Transform(BWT)技術所作的壓縮程式；而 PPMd [12]對原始的 PPM 演算法作改進，其壓縮率比原來的 PPM 好，我們取其最新版本的程式來作測試。PPMd 程式的剖析處理，是直接以 byte 為 token，且其預設的最大階數為 4 階。本文的混合式模型壓縮程式與其它三個壓縮程式的壓縮率實驗數值，如表 19 所示。

表 19　不同壓縮程式之壓縮率

檔案	GZIP	Bzip2	PPMd	混合式模型
達文西	318,647	246,612	227,622	**220,823**
絕代雙嬌	810,977	631,636	554,127	**551,568**
笑傲江湖	1,107,023	875,294	755,819	**744,136**
雜誌	1,861,904	1,449,100	1,302,453	**1,224,959**
現代小說	2,099,860	1,650,032	1,458,279	**1,454,689**
古典小說	2,576,015	2,086,600	1,893,863	**1,873,546**
新聞	2,768,991	2,037,684	1,814,804	**1,687,934**
中文總計	11,543,417 (51.89%)	8,976,958 (40.35%)	8,006,967 (35.99%)	**7,757,655 (34.87%)**
Women	403,661	291,641	**267,491**	276,876
Anne	757,285	547,121	**496,839**	499,203
Ring	1,042,596	752,453	692,930	**680,244**
Bible	1,324,867	929,562	891,492	**878,372**
Harry	2,037,549	1,445,463	1,338,536	**1,307,066**
英文總計	5,565,958 (34.92%)	3,966,240 (24.88%)	3,687,288 (23.13%)	**3,641,761 (22.84%)**

實驗結果顯示，我們的壓縮程式在中文檔案方面，壓縮率都比其他的壓縮程式好，詳細數值是，我們的程式比 PPMd 好 1.12%，比 bzip2 程式好 5.48%，比 GZIP 程式好 17.02%。在英文檔案方面，本文程式所得到的壓縮率，比 GZIP 及 bzip2 都還小，但是與 PPMd 比較時，在英文檔案較小時，壓縮率會比 PPMd 差，當檔案較大時可得到比 PPMd 低的壓縮率，整體來說，本文的壓縮程式，比 PPMd 好 0.29%，比 bzip2 好 2.04%，比 GZIP 好 12.08%。

在速度方面，各個壓縮程式的壓縮、解壓縮速度之量測值，如表 20 所示。量測的方式是，將所有中文測試檔合併為一個大的檔案，英文測試檔也是先作合併，然後以這兩個較大的檔案來作速度測試之實驗。

表 20　各壓縮程式的壓縮、解壓縮速度(Kbytes/s)

檔案		GZIP	bzip2	PPMd	混合模型
中文	壓縮	2397	1810	1782	170.26
	解壓縮	4370	4344	1642	132.37
英文	壓縮	2478	1893	5654	297.28
	解壓縮	5189	4864	4814	217.56

由表 20 可以看出，我們的程式比別的壓縮程式慢，主要的原因是我們的程式在建構、維持高階模型時花了較多時間，並且我們的程式並未作程式效率的最佳化調整。

5. 結論

本論文研究了基於大字符集之詞彙式文本資料壓縮方法，此方法將中、英文資料以詞彙為單位剖析出 token，並在模式步驟中，以高階的詞符集及高階的字符集估算各個 token 的出現機率，接著在編碼步驟將求出的機率以算術編碼轉換為二進位碼，然後輸出到檔案中。我們已將上述流程製作成可實際壓縮、解壓縮的程式，並對此程式作壓縮率的實驗，實驗結果顯示，本論文實作的壓縮程式，對中、英文測試檔壓縮所得到的壓縮率，比同樣使用預測式編碼的壓縮程式 PPMd、及以 BWT 為基礎的 bzip2 程式、及以 LZ77 為基礎的 GZIP 程式都還好。對中文檔案來說，我們的程式得到的壓縮率，比 PPMd 程式好 1.12%，比 bzip2 程式好 5.48%，比 GZIP 程式好 17.02%。對英文檔案來說，

我們的程式比 PPMd 好 0.29%，比 bzip2 程式好 2.04%，比 GZIP 好 12.08%。因此本論文研究之壓縮方法，對中文或英文文章，皆能得到不錯的壓縮效果。

在剖析方面，和前人的研究不同的地方是，我們同時考慮了三種詞彙(中文、英文、符號)的剖析，且每種詞彙的長度皆不等長，因此剖析出的詞符集特別龐大。在中文詞彙方面，我們嘗試剖析出中文一字詞及中文二字詞，若加入中文二字詞的剖析，對某些中文檔案有利，但其改進不大。而在英文詞彙方面，我們嘗試剖析長度不超過 20 的英文詞彙之方式。在模式(modeling)方面，本論文嘗試以部分匹配之預測模型及混合式模型兩種作法來估算詞彙的機率。和前人研究的不同點是，本論文實作較龐大、高階的模型結構，主要是想研究此作法對壓縮率及壓縮時間的影響。實驗結果顯示，以混合式模型來預測詞彙機率，對中、英文檔案都能得到不錯的壓縮率。

為了能夠支援大字符集情況的壓縮處理，在詞符集及字符集部分，我們以支援累積之二元搜尋樹的資料結構來建構，並儲存各詞彙、字元之出現次數。在編碼方面，本論文採用算術編碼，但由於本論文是使用大字符集之預測模型，因此我們必須對算術編碼程式作一些修改，以支援大字符集符號之編碼處理。

未來可改進的方向，在剖析方面，可以考慮更多種語言的特性，這樣對其他語言的文本資料，應能像本論文對中、英文文本資料之壓縮一樣，得到比一般壓縮程式還要好的壓縮率。而在支援累積之二元搜尋樹的資料結構方面，如果可以在二元樹中作加入、刪除資料時，隨時維持所有二元樹為平衡樹，如此在壓縮、解壓縮的速度方面，應能有更好的效率。另外在混合式模型之實作中，也可以研究是否有更合適的權重計算方法及逃脫機率計算方法。

參考文獻

[1] Huffman, D. A., “A Method for the Construction of Minimum Redundan-cy Codes”, Proc. Inst. Electr. Radio Eng., 40(9), pp. 1098-1101 (1952).

[2] Rissanen, J. J., “Generalized Kraft Inequality and Arithmetic Coding”, IBM J. Res. Dev., 20, pp. 198-203 (1976).

[3] Ziv, J. and A. Lempel, “A universal Alg- orithm for Sequential Data Com- pression”, IEEE trans., Information Theory, 23(3), pp. 337- 343 (1977).

[4] Ziv, J. and A. Lempel, “Compression of individual sequences via variable-rate coding”, IEEE trans., Information Theory, 24(5), pp. 530-536 (1978).

[5] Welch, T. A., “A technique for high- performance data compression”, IEEE Computer, 17(6), pp. 8-19 (1984).

[6] Sayood, K., Introduction to Data Compression, 2’nd ed., Morgan Kauf- mann (2000).

[7] Witten, I. H. and T. C. Bell, “The zero- frequency problem: estimating the probabilities of novel events in adaptive text compression”, IEEE trans. Information Theory, 37(4), pp. 1085-1094 (1991).

[8] Moffat, A., R. Neal, and I. H. Witten, ” Arithmetic Coding Revisited”, Proc. of Data Compression Conference, DCC'95, pp. 202-211 (1995).

[9] Vines, P. and J. Zobel, “Compression Techniques for Chinese Text”, Software Practice and Experience, 28(12), pp. 1299-1314 (1998).

[10] Gu, H. Y., “A Large-Alphabet-Oriented Scheme for Chinese and English Text Compression”, Software: Practice and Experience, 35(11), pp. 1027-1039 (2005).

[11] Cleary, J. G. and I. H. Witten, “Data Com- pression using Adaptive Coding and Partial String Matching”, IEEE Trans. Communication, 32(4), pp. 396-402 (1984).

[12] Shkarin, D., “PPM: One Step To Practicality”, Proc. of Data Compression Conference (UT), pp. 202- 211 (2002). Available at: http://compression.graphicon.ru/ds/.

[13] Horspool, R. N. and G. V. Cormack, “Constructing Word-Based Text Comp- ression Algorithms”, Proc. of Data Compression Conference, DCC'92, pp. 62-71 (1992).

[14] Cheng, K.-S., G. H. Young, and K.-F. Wong, "A Study on Word-Based and Integral-Bit Chinese Text Compression Algorithms", Journal of the American Society for Information Science, 50(3), pp. 218-228 (1999).

[15] 劉景民，基於大字符集柏洛菲勒轉換之中文文本資料壓縮方法，碩士論文，台灣科技大學電機工程研究所，2004。

[16] Microsoft Windows Codepage 950, http://www.microsoft.com/globaldev/reference/dbcs/950.mspx.

[17] Visual C# Developer Center, “An Extensive Examination of Data Stru- ctures Part 2: The Queue, Stack, and Hashtable”, http://msdn.microsoft.com/vcsharp/default.aspx?pull=/library/en-us/ dv_vstechart/html/ datastructures_guide2. asp.

[18] Nelson, M., “Arithmetic Coding + Statistical Modeling = Data Compress- ion”, http://www.dogma.net/markn/ articles/arith/part1.htm.

[19] Fenwick, P. M., “A New Data Structure for Cumulative Probability Tables”, Software: Practice and Experience, 24, pp. 327-336 (1994)

[20] Moffat, A., “Implementing the PPM Data Compression Scheme”, IEEE trans. Communications, 38, pp. 1917-1921 (1990)

[21] Belll, T.C., J. G. Cleary, and I. H. Witten, Text Compression, Prentice Hall Inc., Englewood Cliffs, New Jersey (1990)

[22] GZIP source code, ftp://prep.ai.mit.edu/pub/gnu/gzip/

[23] Bzip2 home page, http://www.bzip.org