基於大字符集柏洛菲勒轉換之中文文本資料壓縮方法

A Chinese Text Compression Scheme Based on Large-Alphabet BW-Transform

古鴻炎
Hung-Yan Gu
國立台灣科技大學資工系
guhy@mail.ntust.edu.tw

劉景民
Jin-Min Liu
國立台灣科技大學電機所
M9007313@mail.ntust.edu.tw

摘要

本論文提出一種基於大字符集柏洛-菲勒轉換(Burrows-Wheelers Transform, BWT) 之中文文本資料的壓縮方法，先以 Big-5 加上 ASCII 形成的大字符集(alphabet)來剖析輸入的中文文字檔案，再接著進行 BWT、MTF(Move to Front)、和算術編碼的處裡。我們也研究了，在大字符集要求下能夠適用於 BWT、MTF 和算術編碼處裡上的實作方法，以提升處理的速度。我們已經將這個壓縮方法製作成可以實際使用之軟體程式，對於中文文字檔案的測試實驗，結果顯示我們方法獲得的壓縮率，比一般常被使用的 Win-ZIP 好約 12%，比 Win-RAR 好約 4%，而比原始的基於 BWT 的壓縮軟體 BZIP2 的壓縮率好約 1%~2%。

關鍵詞：文本資料壓縮、大字符集、BWT 文字轉換、算術編碼

1. 前言

隨著電腦逐漸被普遍的使用，電腦所需要儲存的資訊越來越多，如何有效率的儲存資料，已經變成了一個相當重要的課題，而資料壓縮就是解決這個問題的方法。關於文本資料壓縮編碼的問題，過去被提出的解決方法可分成兩類[1][2]，一類為詞典式編碼(Dictionary Coding)，如 LZ77、LZ78、LZSS 等方法，另一類為預測式編碼(Predictive Coding)，如 PPM(Prediction with Partial Match)、Dynamic Markov Compression 等。一般而言，這兩類的壓縮方法在速度跟壓縮率上是各有其優缺點的，詞典式編碼的特點是編解碼速度相當快，但是壓縮率較差一些，而預測式編碼則是壓縮率則是相當不錯，但是處理速度較慢，記憶空間花費較多。

在 1994 年，Burrows 和 Wheeler 兩位學者提出了一種非失真性的文字轉換方法[2] ，稱為柏洛-菲勒轉換(BWT)，這種轉換是將輸入的文字去做區塊排序(Block Sorting)，然後可得到一種特徵明顯的符號序列，即此序列會變成是由一連串的相同符號之片段所串接成的符號序列，並且具有一種區域性參考(local reference)的特性，而這種區域性參考的特性，後續可以使用 MTF 轉換處理來降低文章的熵值(Entropy)，之後再做可變碼長編碼，即可達到相當不錯的壓縮效率。以 BWT 為基礎的壓縮方法，具有近乎預測式編碼的壓縮效率，而壓縮速度則稍遜於詞典式編碼，目前比較具有代表性的如 BZIP、BZIP2 等壓縮程式[3]。可是，這些軟體都是採用 8-bits ASCII 之小字符集來對輸入檔案作剖析(parsing)處理，目前文獻上缺乏採取大字符集剖析的 BWT 壓縮法的研究，因此我們在此篇論文的研究中，依據 BWT 的文字轉換觀念，嘗試以大字符集之 BWT 處理來建構一種配合中文文字特性的壓縮方法。本文第二節將說明我們所研究的壓縮方法的處理步驟；第三節討論測試實驗的結果；第四節則是結論。

2. 大字符集 BWT 之壓縮方法

本論文提出的壓縮方法之流程如 Fig.-1 所示，第一個步驟是，採取以 Big-5 加上 8-bits ASCII 所形成的大字符集來剖析輸入的中文文字檔案；第二個步驟是，進行大字符集之 BWT 處理；第三個步驟是，以改進的 Heap MTF 方法，來進行熵值減低的處理；第四個步驟是，以修正的、兼顧速度與壓縮效率的的算術編碼方法，來進行編碼處理。各步驟的詳細的處理程序，將在以下的各子節中說明。

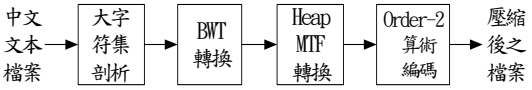

Fig.-1. 大字符集 BWT 壓縮方法的處理步驟

2.1 大字符集之剖析

我們允許輸入的文本資料檔案裡，可以有中、英文夾雜之情況，因此這裡採取以 Big-5 碼加上 8-bits ASCII 碼來形成一個大字符集，以對輸入的文字檔案進行剖析處理，將各個中、英文字符號切割出來，實作上程式內部以一個 2-bytes 之整數來代表各個符號。Big-5 中文碼為一種雙位元組(Byte)的編碼方式，能夠和 7 bits 的 ASCII 碼相容並存，Big-5 中文碼的編碼範圍是，第一個位元組範圍介於 161~254 之間，第二個位元組的範圍則分別落於 64~126 與 161~254 的兩段範圍。所以，這裡採取的大字符集總共有 14756 (Big5 部分) + 256 (ASCII 部分) = 15014 個字符，如 Fig-2 所示。

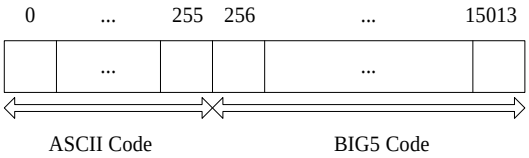

Fig-2. ASCII 碼與 BIG-5 碼合成之大字符集

這裡為什麼要把一個中文字碼的兩個位元組當作一個單位來作處理，而不直接當作兩個位元組來分別作處理？因為我們經過實驗，發現中文字碼以雙 byte 為一個單位的處理方式，比起單 byte 處理方式要好上 3%左右的壓縮率，實驗結果的數據在 3.1 節列出。

2.2 BWT 演算法

為了方便介紹 BWT 演算法，我們使用文獻裡的一個範例[4]，假設輸入字串 $S='abraca'$，長度 $N=6$，且字符集 $A=\{'a','b','c','r'\}$。

BWT 演算法的步驟如下：

1. 將字串 S 每次往左移動一格，並且移動 N 次，形成一個概念性的矩陣 M。

$$M = \begin{bmatrix} a & b & r & a & c & a \\ b & r & a & c & a & a \\ r & a & c & a & a & b \\ a & c & a & a & b & r \\ c & a & a & b & r & a \\ a & a & b & r & a & c \end{bmatrix}$$

2. 將 M 以列(row)為單位作字元順序(lexical order)的字串排序，得到 M'。此外，必須在排序之後，紀錄原本字串 S 出現在 M' 矩陣中的第幾列(row)，列編號或陣列下標都從 0 編起。在此我們設是 I。

$$M' = \begin{bmatrix} a & a & b & r & a & c \\ a & b & r & a & c & a \\ a & c & a & a & b & r \\ b & r & a & c & a & a \\ c & a & a & b & r & a \\ r & a & c & a & a & b \end{bmatrix}, I = 1$$

3. 第二步驟做完之後，可得到一個矩陣 M' 以及一個 $I=1$，而 BWT 轉換後的字串 L 即是 M' 矩陣中的最後一欄(column)的元素所串接而成。所以得到 $L='caraab'$ 以及一個 $I=1$。因此 BWT 處理後會送出 L 和 I，解碼端可依據這兩個資訊，來正確的將原始字串 S 反向轉換回來。

反向轉換的作法，可以參考 Burrows 和 Wheeler 所發表的論文[4]，裡面有詳細的解說，大體上是先對得到的 L 作排序以得到 F，即矩陣 M' 的第一欄的資訊。再利用 L、F 兩者來建造一個對映(mapping)用的陣列 T，之後可根據 I 及 T 來解出 S。

在前面 BWT 演算法的介紹裡，雖然說使用的資料結構是 NxN 的矩陣，但是實際上因為陣列中每個列都只是原始字串做數次旋轉(rotation)後的結果，所以只要一個長度為 $2N$ 的陣列就可以代表整個矩陣，所以空間複雜度只需 $O(n)$ 即可。另一個要考量的是，在正向轉換時必須使用到字元順序的字串排序，初步來看，可以使用 n 次的排序處理來分別對各欄作字元排序，所以時間複雜度為 $O(n^2 \log n)$，實作上，我們使用了 Bentley 和 Sedgewick 所提出的 ternary partition 的 quick sort[5]來做字串排序，用此方法可把排序的時間複雜度降為 $O(n^2)$。

經過 BWT 轉換之後，我們可以觀察輸出字串 L 的特性。因為 L 經過排序後，可以得到矩陣 M' 的第一欄 F，所以 L 有相對應的第一欄 F 來當作參考。我們可以把 F 視為由一連串有次序(order)的字符區段(segment)所組成，而 L 的各個元素字元在原始字串 S 中，都是對應的 F 的元素的前一個字元，所以我們可將 L 依照 F 的區段分割成許多具有區域關連性的區段。舉例來說，假設在一篇中文文章中，有許多「你們」、「我們」、「他們」等等以「們」為結尾的詞語，則在經過 BWT 正轉換之後，F 中「們」的區段相對應的 L 區段，就會由許多的「你」、「我」、「他」所組成，所以在這個

區段中，連續參考「你」、「我」、「他」的機率相對於其他的區段來說，可以說是相對的高很多。

2.3 Heap MTF 轉換

為了能夠針對 BWT 輸出字串所具有的區域性參考的特性來作進一步的處理，以降低熵值，我們必須要找出一種能夠快速調適各個區段的區域特性的處理方法。一般而言，文獻上提出的基於 BWT 之壓縮方法，都是在這個階段使用 MTF 轉換來做調適處理。MTF 的觀念是，以機動移動各字元在字符集中的位置，及間接對位置值作編碼(而不直接對字元本身作編碼)，來得到調適的效果，使得熵值降低。在此先介紹原始的 MTF 作法，假設一開始字符集為 $A=\{'a', 'b', \ldots, 'x', \ldots.\}$，且其元素的排列是有次序性的，若本次碰到的一個輸入的字元是'*x*'，則先送出'*x*'在字符集的位置值 *j*（假設 *A*[*j*]= '*x*'），然後再把'*x*'移到字符集 *A* 的最前端，使得 $A=\{'x', 'a', 'b', \ldots\}$。後續的字元編碼方式，也是先送出所碰到的輸入字元在字符集 *A* 裡的位置值，然後再將該字元移動到字符集的最前頭的方式來進行，因此若字元'*x*'緊接著再次被參考到，則送出的位置值就是 0 了。

經由這種 MTF 的處理，在某個 BWT 輸出的 *L* 字串的一個區段的開頭，即可以把該區段裡常用的字符移動到字符集的前端，而該區段的中段和後段，大部分都只參考到字符集的第 0、第 1 個位置，或者是字符集較前端的位置，如此使得 MTF 轉換輸出的字符集字元的位置值序列，會形成數值小者，出現的機率較高的情形，因此若先進行 MTF 轉換，再對輸出的位置值作編碼，就可以大為減少 BWT 輸出字串的熵值。

可是實作上，因為 MTF 每次參考到一個符號，最多必須移動 *N*-1 個字符(設 *N* 為字符集的元素個數)，這在 7-bits 的 ASCII 小字符集中，一次的字符集參考最多只會移動 127 的字符，造成的處理速度減慢並不會很大，但是在大字符集中，因為字符集元素數量巨大(15014 個)，若每次參考到一個字符，就要作一萬多個字符的移動，在處理上是相當費時的。所以我們要針對大字符集的情況，來設計一種兼顧處理速度的 MTF 變形作法。

在本論文中，我們研究了一種 heap 樹結構的變形，這種作法先把 0～15013 字符集內的字符，分別分配到 13 顆 heap 樹內，如 Fig-3 所示，每一顆樹以 $T_0, T_1, \ldots, T_{12}$ 編號之，且第 i

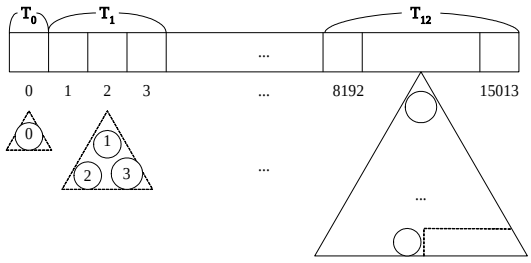

Fig-3. 字符集元素分割成 13 棵 Heap 樹之結構

顆樹的大小為 $2^{i+1}-1$，如此每當一個字符被參考到的時候，就可以依 heap 結構中的父子關係形成的路徑，沿著字符的父親的方向，移到樹的樹根(root)，而當移到一棵樹的樹根之後，就檢查是否已在 T_0，若不是則再往前一棵樹的樹葉(leaf)節點移動。一個重要的課題是，要挑選前一棵樹的那一個葉節點做交換，才不會影響壓縮率？一開始我們使用的策略是跟前一棵樹的最右邊的節點做交換，可是這樣會導致一個問題，就是交換到上一棵樹之後，會一直沿著 heap 樹的最右邊那條路徑一直上去，導致於某些地方（如樹的中左邊）形同虛設，為了避免這個問題，後來我們改成採取輪流式的策略來做交換，亦即先對前一棵樹的最右邊的葉節點作交換，下次交換時則是與右邊往左的第二個葉節點做交換，以此類推。如此的作法，可以避免造成某些地方的字符集元素不會被移動到，而造成壓縮率的降低。

前述的 heap 樹之 MTF 作法，速度上可將複雜度 $O(n^2)$ 改善成 $O(n\log n)$，而壓縮率方面，由實驗驗證，比起原始版本的 MTF，相差的程度並不多，詳細的實驗數據在 3.2 節中列出。

MTF 也有許多變形的作法，在本論文中，我們分別實驗了 MTF-1[6]、MTF-2[7]、MTF-halfway[8]對於壓縮率所造成的影響。

在 MTF-1[6]作法裡，若本次遇到的輸入字元是'*x*'，則先送出'*x*'在字符集的位置值 *j*（假設 *A*[*j*]='*x*'），然後考慮兩種情況，第一種是 $j>1$ 時，則我們將'*x*'移動到字符集的第一個位置（也就是 T_1 的樹根），使得 $A=\{'a','x','b',\ldots\}$。另一種情形是 $j\le 1$ 時，則我們將'*x*'移動到字符集第 0 個位置(也就是 T_0)，使得 $A=\{'x','a',\ldots\}$。這種變形方法使得調適的速度減慢了一步，位置值大於 1 的值必須連續兩次被參考到，才會移動到 0，這樣的作法可以避免一個不常出現的符號，也被移動到第 0 個位置，導致於其它常用符號的位置後退了一格，而使得壓縮率下降。

另外一種變形作法 MTF-2[7]沿襲了

MTF-1 的作法，只是當 $j \le 1$ 時，再加上一個判斷條件，就是前一個已編碼的符號必須是 0，才會將符號'x'移至第 0 個位置。這種作法，更進一步的減少在原始版本之 MTF 中，快速調適所帶來的缺點。

MTF-halfway[8]是另外一種調適速度更慢的作法，假設一個符號'x'(位於 T_i)被參考到之後，下一步則是往前移動到 $T_{\lfloor i/2 \rfloor}$ 這群的樹根。這種調適的速度比起 MTF-1 和 MTF-2 的更慢。但是因為字符移動的次數變少，所以轉換速度也會相對的變快。

以上提到的三種變形策略，其壓縮率的實驗，列在 3.3 節。由實驗結果我們發現，MTF-halfway 這種調適速度較慢的方式，對於中文之大字符集，會有較佳的壓縮率。

2.4 算術編碼

經由前一階段的 Heap MTF 轉換之後，可以把 BWT 輸出的 L 字串，轉換成為具有低熵值的字符集位置值之序列，接著我們可對這個序列作可變長度碼的編碼。一般而言，可變長度碼之編碼可分為兩大類，一類為霍夫曼編碼(Huffman Coding)，另一類為算術編碼(Arithmetic Coding)[1][2]。霍夫曼編碼以 Greedy 策略，根據文章內所有符號的出現機率，建立出一棵具有最小成本的編碼樹(code tree)，而當要作編碼的時候，就由樹根(root)向下走訪到所要編碼的符號，而每走一個邊(edge)，就送出一個位元(bit)，由此可知霍夫慢編碼是一種整數位元數的編碼作法。

另外一種編碼方法是算術編碼，它跟霍夫曼編碼不一樣的地方是，算術編碼在觀念上可以用分數個 bit 來表示一個符號。一開始編碼時，算術編碼使用一個半開之區間[0~1)，然後根據各個字符的預估出現機率值來分割此區間，當碰到某一個輸入之符號，就將原先的區間窄化成該輸入符號所分配、對應的區間。接著再依所估計出的字符之出現機率，去分割此窄化過的區間給各個可能的字符，再由下一個輸入之符號，來決定區間要再窄化成那一個子區間。如此繼續下去等到編碼完後，區間會變成非常窄，左右邊界的數值會有一連串的前導小數是共同的，只要送出這串共同的小數數目之字串，即可解回原先的符號字串，更為詳細的介紹請參考相關文獻[10]。

在這裡我們必須決定採取其中一種的可變長度編碼方法，由於我們觀察 MTF 輸出的位置值序列，發現位置值 0 佔了大部分的機率，常常超過 1/2，對於這種非均勻(Non-uniform)的機率分佈，使用霍夫曼編碼之壓縮效率不會很好，因為霍夫曼編碼，只能對機率值為 $(1/2)^n$ 的字符，作沒有損失編碼空間的編碼，也就是對 MTF 輸出的 0 值，最少也需花費 1 個位元來表示，並不能以小於 1 個位元以下的分數個位元來表示，所以會造成壓縮效率不會最好。因此我們決定採用可作分數個位元編碼的算術編碼方法。

不過算術編碼也有其缺點，當對大字符集裡的字符做編碼時，所使用的機率估計模型，模型的調適會需要大量的計算，此外要記錄過去歷史中各符號的出現次數，也要消耗不少的記憶容量，因此我們認為符號(或位置值)分群(Grouping)的觀念是必要的[2][9]，在此我們將 Heap MTF 轉換輸出的可能位置數值，分成 14 群，詳細分法為，位置值 0 分成一群、位置值 1 分成一群，而其它群含蓋的位置值範圍如 Table-1 所示。當我們要對一個位置值 x 作編碼時，首先使用下列的公式來將其分成群碼 G 與元素碼 E：

$$G = \lceil \log_2(x+1) \rceil \tag{1}$$

$$\begin{cases} E = 0, & if \quad G \le 1 \\ E = x - 2^{G-1}, & if \quad G > 1 \end{cases} \tag{2}$$

其中元素碼所佔的位元個數為 G-1 若 G > 1。在此我們只對群碼 G 採取算術編碼的方法來進行編碼，而對於元素碼 E，則是送出 G-1 個位元之固定碼長的編碼當作輸出。

Table-1. MTF 輸出位置值之分群

群碼	含蓋的位置值
0	0
1	1
2	2、3
3	4、5、6、7
…	…

Isal 等人在 2002 年所提出的論文[8]，指出 MTF 輸出的位置數值，並不一定全是隨機分佈，而是仍然具有一定程度的相關性可被利用，該篇論文提出了 CINT 系列的算術編碼方法來作編碼。我們在本論文的研究過程裡，發現到分群後的群碼也具有一定程度的相關性，所以我們就採用了 Markov 模型來移除這些相關性，並且實驗何種階數(order)的模型才能得到最好的壓縮率。

在對群碼 G 作算術編碼的實作方面，我們採用了 Mark Nelson 所提供的 open source

Table-2. 字符集大小對壓縮率的影響

	哈利波特	紅樓夢	窗外	青青河邊草	一廉幽夢	神雕俠侶
原始大小	1,277,266	1,425,526	352,122	209,911	272,192	1,717,819
小字符集	531,387	675,073	149,583	94,149	111,070	809,057
大字符集	491,204	629,953	142,549	89,543	105,716	743,282

Table-3. MTF 與 Heap MTF 處理時間比較

	哈利波特	紅樓夢	窗外	青青河邊草	一廉幽夢	神雕俠侶
MTF	5.70(sec.)	8.63(sec.)	1.83(sec.)	1.29(sec.)	1.38(sec.)	9.94(sec.)
Heap MTF	1.26(sec.)	1.65(sec.)	0.36(sec.)	0.23(sec.)	0.26(sec.)	1.97(sec.)

Table-4. MTF 與 Heap MTF 壓縮大小比較

	哈利波特	紅樓夢	窗外	青青河邊草	一廉幽夢	神雕俠侶
原始大小	1,277,266	1,425,526	352,122	209,911	272,192	1,717,819
MTF	498,213	643,439	143,965	90,171	106,412	751,630
Heap MTF	499,466	644,932	144,327	90,410	106,714	753,796

的算術編碼程式[10]來製作我們的壓縮軟體。關於數種階數(order)的測試，階數是表示在作編碼時，一個群碼值的出現機率的計算方式會有所不同，例如當要對 k 時刻的群碼 x_k 做編碼，若階數為 1 時，則必須估計前一個時刻群碼為 x_{k-1} 的條件下，發生 x_k 的機率 $P(x_k \mid x_{k-1})$，然後用此機率估計值來作算術編碼。而當在階數 2 時，則必須估計 $x_{k-2}\ x_{k-1}$ 的條件下，發生 x_k 的機率值 $P(x_k \mid x_{k-2}x_{k-1})$。3.4 節中列出了我們使用各種階數去做測試的實驗結果。

3. 測試實驗之結果與討論

這裡我們採用了六篇中文的長篇小說以及短篇抒情短文來作為壓縮率與速度測試之檔案，它們分別是翻譯小說哈利波特、中文古典小說紅樓夢、瓊瑤的三篇短篇抒情小說窗外、青青河邊草、一廉幽夢，和金庸的武俠小說神鵰俠侶。每次處理的 Buffer 單位為 1Mb，亦即 524,288 個雙位元的資料。3.1 小節我們實驗了大字符集和小字符集為處理單元的壓縮率比較；3.2 小節實驗原始 MTF 與本論文提出的 Heap MTF 以及其它四種變形方式在壓縮率及處理時間上的比較；3.3 小節則實驗了各種階數的算術編碼之壓縮率比較。在 3.4 小節，我們針對壓縮率以及壓縮時間，來與其他的軟體做比較。

3.1 大字符集與小字符集壓縮率比較

在 2.1 節我們介紹了大字符集的剖析方式，本小節就實驗在相同的最佳處理機制下(BWT + Heap MTF-halfway + Order-2 算術編碼)，大字符集與小字符集對壓縮率的影響。Table-2 列出單 Byte 與雙 Byte 為單位作處理的壓縮結果。由此表可以發現，有沒有先做大字符的剖析，對於壓縮率，具有一定程度的影響。所以整體來說，以大字符集來剖析中文，比較符合中文文章的特性，而能得到較好的壓縮率。

3.2 MTF 與 Heap MTF 的壓縮率比較

在此，我們針對在 2.3 小節所提出以 Heap 樹的架構來作 MTF 的方法，來與原始版本的 MTF 作比較。一開始，我們設計 Heap MTF 這個方法，就是要提高大字符集在做 MTF 時的處理速度，Table-3 列出了兩者處理時間的比較，我們可以發現，這樣的處理方式，果然比起原始版本的 MTF，在處理速度方面，有很大的進步。如果更進一步的探討，其實 MTF 與 Heap MTF，基本精神並沒有改變，就是參考到一個符號之後，馬上移動到字符集前面的位置，其差別只是在於，要將參考到的符號，移動到字符集前面的什麼位置。MTF 的處理是做線性移動，而 Heap MTF 是沿著最底層的樹，開始往上走到最上層的一棵樹。當參考一個後面位置的字符時，原始之 MTF 會把全部之前的字符往後移一個位置，而 Heap MTF 則是沿著 heap 的某條路徑，走到字符集的開頭，並且我們依 2.3 節提及的輪流式交換的處理，使之前的字符會以平均的方式，往字符集的後面移動，所以當字符集一大的時候，這兩者的處理速度差別就顯現出來了。

接著，比較兩者在壓縮率方面的差別。由 Table-4 可發現，Heap MTF 與 MTF 方法的壓縮率相差並不多，所以使用 Heap MTF 的方法是可行的。

Table-5. 各種 MTF 變形方法的壓縮率比較

	哈利波特	紅樓夢	窗外	青青河邊草	一廉幽夢	神雕俠侶
原始大小	1,277,266	1,425,526	352,122	209,911	272,192	1,717,819
MTF	499,466	644,932	144,317	90,379	106,697	753,202
MTF-1	496,992	641,373	143,764	90,162	106,332	750,083
MTF-2	496,666	640,942	143,653	90,136	106,291	749,690
MTF-halfway	491,203	629,952	142,548	89,542	105,715	743,281

Table-6. 各種階數之算術編碼壓縮大小比較

	哈利波特	紅樓夢	窗外	青青河邊草	一廉幽夢	神雕俠侶
原始大小	1,277,266	1,425,526	352,122	209,911	272,192	1,717,819
Order-0	498,570	635,428	144,052	90,389	106,863	751,774
Order-1	491,753	630,345	142,338	89,209	105,440	743,507
Order-2	491,203	629,952	142,548	89,542	105,715	743,281
Order-3	500,303	641,452	146,738	92,881	109,268	755,159

3.3 MTF 及其變形方法的壓縮率比較

這裡我們將就把 2.3 節所提及的 MTF-1、MTF-2、MTF-halfway、MTF-queue 等四種方式以 Heap 樹的架構實作，並作壓縮率的比較。Table-5 列出了各種變形方法壓縮率的比較。在此我們發現比較慢的調適方式 MTF-halfway 比較適合中文 BWT 轉換後的處理。原因為中文大符集符號的分佈比較廣泛，有許多中文字只被參考到一次，假如將參考到一次的符號馬上移動到字符集的前端，因為以後不會被參考到，所以會使其它出現頻率較高的符號位置會被後退一格，這樣將會導致壓縮率變差。而調適得太慢也無法適應 BWT 文字序列的特性，所以在本論文中，發現以 MTF-halfway，所得出來的壓縮結果，比起其它方式，是比較好的選擇。

3.4 各種階數算術編碼壓縮率之實驗

在 2.4 節我們介紹了不同階數之算術編碼的作法，因此這裡就進行壓縮率好壞的比較，結果得到的數據如 Table-6 所示，我們發現當檔案大的時候，使用 2 階的壓縮率會比較好，當檔案小的時候，使用 1 階的效果會比較好，不過和 2 階的壓縮率差不多，而 3 階以上，壓縮率就開始變差了。所以根據實驗可以發現，2.4 節所介紹的分群觀念，群碼在 2 階時的條件機率，還是具有一定程度的相關性，但到 3 階，資料已經跟 0 階一樣變得較為隨機分佈了。

3.5 與其他壓縮法之壓縮率與壓縮時間比較

Table-7 為我們的方法與其他壓縮方法的壓縮率比較，這裡我們找了 Bzip2[3]、Win-ZIP、Win-RAR，以及本文作者之一今年初再改進的 LZGD 壓縮法[9]來做比較，其中 Bzip2 是屬於 BWT 為基礎的壓縮方法，而 Win-ZIP、LZGD 則是屬於辭典式編碼的作法，且 LZGD 是一種專門針對中、英文文本資料的大字符集之壓縮方法。實驗得到壓縮率的結果，本論文的方法比起 BZIP2，好了將近有 1%~2%的效率，在此壓縮率的定義，指的是壓縮後大小除以原始大小的比值。另外，比起 LZGD，好了有 1%左右的效率。至於其他 WinZIP 以及 WinRAR，由於這兩套軟體是商業軟體，強調的是速度，所以本論文方法的壓縮率比起這兩套軟體明顯的好上許多。

在壓縮時間的實驗上，我們使用了 Pentium 4 1.6G 的 PC 來做實驗，在此我們將參與實驗的六個中文檔案串接在一起，總共為 5.011Mb，然後用以測試壓縮時間和解壓縮時間，並且比較平均壓縮 1Mb 資料所要花的秒數。實驗的結果如 Table-8 所示，目前本論文方法的速度較慢，原因之一是方法的複雜度較高，另一原因是程式尚未作最佳化之考慮。

4. 結論

我們提出了一個基於大字符集 BWT 的中文資料壓縮方法，該壓縮法由大字符集剖析、BWT、Heap MTF、和群碼算數編碼等四個處裡步驟組成。由實驗結果得知本論文的方法可得到最好的壓縮率。此外，對於 MTF 之步驟，我們提出以 Heap 樹結構，來改進大字符集 MTF 處理的速度。再者，對於 MTF 輸出的位置值，我們經由實驗發現，群碼部分可以用兩階的機率估計模型來作算術編碼的處理，以獲得較佳的壓縮率。不過在執行速度上，仍比其他市面上的軟體差許多。未來可以再改善的方

Table-7. 與其他壓縮法壓縮率之比較表

	哈利波特	紅樓夢	窗外	青青河邊草	一廉幽夢	神雕俠侶	平均壓縮率
原始大小	1,277,266	1,425,526	352,122	209,911	272,192	1,717,819	5,254,836
本論文壓縮法(BLA)							
壓縮後	491,203	629,952	142,548	89,542	105,715	743,281	2,202,241
壓縮比	38.4%	44.2%	40.5%	42.6%	38.8%	43.3%	41.9%
BZIP2							
壓縮後	513,806	662,841	148,096	92,709	109,789	775,289	2,302,530
壓縮比	40.2%	46.5%	42.1%	44.2%	40.3%	45.1%	43.8%
Win-ZIP							
壓縮後	662,223	814,095	185,542	112,702	137,640	986,867	2,899,069
壓縮比	51.8%	57.1%	52.7%	53.7%	50.6%	57.4%	55.2%
Win-RAR							
壓縮後	579,279	729,510	163,511	103,601	121,515	849,103	2,453,278
壓縮比	45.3%	51.2%	46.4%	49.3%	44.6%	49.4%	46.7%
LZGD							
壓縮後	509,417	647,292	145,711	90,141	108,188	760,911	2,261,660
壓縮比	39.9%	45.4%	41.4%	43.0%	39.7%	44.3%	43.1%

Tabl-8. 壓縮時間與解壓縮時間比較表

	本論文方法 (BLA)	BZIP2	WinZIP	WinRAR	LZGD
壓縮時間	19.059(sec.)	8.24(sec.)	1.45(sec.)	8.40(sec.)	5.09(sec.)
解壓縮時間	12.945(sec.)	5.73(sec.)	0.44(sec.)	0.51(sec.)	1.24(sec.)
1Mb 壓縮時間	3.80(sec.)	1.64(sec.)	0.29(sec.)	1.68(sec.)	1.02(sec.)
1Mb 解壓縮時間	2.58(sec.)	1.14(sec.)	0.09(sec.)	0.10(sec.)	0.25(sec.)

向是，在 BWT 階段使用更快速的排序方法，在 MTF 階段也可以採取其他的 MTF 策略，而在算術編碼階段也可以使用較佳的演算法來改進速度，所以我們以後將朝上述的方向來努力。

參考文獻

[1] T. C. Bell, J. G. Cleary, and I. H. Witten, Text Compression, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1990.

[2] Sayood Khalid, Introduction to Data Compression, 2’nd ed., Morgan Kaufmann, 2000.

[3] Julian Seward, “The bzip2 and libbzip2 homepage”, http://sources.redhat.com/bzip2 .

[4] M. Burrows and D.J. Wheeler, “A Block-sorting Lossless Data Compression Algorithm”, SRC Research Report 124, Digital System Research Center, Palo Alto, USA, May 1994.

[5] J. Bentley and R Sedgewick, “Fast Algorithm for Sorting and Searching String”, Proc 8th ACM-SIAM Symposium on Discrete Algorithms, pp. 360-369, 1997.

[6] B Balkenhol, S Kurts, and Y. M. Shtarkov,” Modifications of the Burrows and Wheeler Data Compression Algorithm.”,Proceedings of the IEEE Data Compression Conference, 1999; 188-197.

[7] B Balkehhol, Y. M. Shtarkov, “One attempt of a compression algorithm using BWT”, Preprint 99-133, SFB343: Discrete Structures in Mathematics, Falculty of Mathematics, University of Bielefeld, 1999.

[8] R. Y. K. Isal, A. Moffat and A. C. H. Ngai, “Enhanced Word-Based Block-Sorting Text Compression”, Twenty-Fifth Australasian Computer Science Conference(ACSC), 2002.

[9] Hung-Yan Gu,“A New Chinese Text Compression Scheme Combining Dictionary Coding and Adaptive Alphabet-character Grouping”, Journal of Computer Processing of Chinese and Oriental Languages, Vol. 10, No.3, pp. 321-335, (1997).

[10] Mark Nelson, “Arithmetic Coding + Statistical Modeling = Data Compression”, http://www.dogma.net/markn/articles/arith/part1.htm.