Modified LZ77 與 LZSS 之編碼格式以減少位元膨脹問題

陳順智	田子坤	蔡明賢
南台科技大學	南台科技大學	南台科技大學
電子工程系副教授	電子工程系副教授	電子工程系碩士班研究生
scchen@mail.stut.edu.tw	tktien@mail.stut.edu.tw	M9730301@webmail.stut.edu.tw

摘要

LZ77 及 LZSS 兩種演算法對於非文字型態的待編碼資料，如果符號的再出現週期過長，超過了預設的搜尋緩衝區大小，位元膨脹問題(bit-expanding problem)就會經常發生，造成該兩種演算法對於此類型資料的壓縮效果不佳。本篇論文特針對此缺點，提出了同時考慮搜尋緩衝區大小及編碼格式的解決方案。實驗結果顯示，修改後的演算法對於 ASCII 與非 ASCII 的資料都可以得到更好的壓縮效果。

關鍵詞：LZ77, LZSS, bit-expanding

Abstract

This paper aims at the drawback of LZ77 and LZSS algorithms to improve the compression rate for the non-ASCII data. For the encoding symbol with the reappearance period longer than the size of the search buffer, the bit-expanding problem is easily occurred and therefore degrades the compression efficiency in both algorithms. In the proposed algorithm, the size of search buffer is well defined and the code word format is modified to solve the bit-expanding problem. Experimental results show that these modifications improve the compression rate for both non-ASCII and ASCII data.

Keywords: LZ77, LZSS, bit-expanding

1. 前言

在高速、低成本之資料傳輸與儲存應用領域中，資料壓縮是計算機科學領域中非常重要的課題之一，它是由資訊理論所提出來的原理所發展出來的，如今它可以應用到非常廣泛的應用上。資料壓縮分無失真壓縮與失真壓縮，而無失真資料壓縮已成為一種關鍵且重要的技術。

無失真資料壓縮之特色乃藉由降低傳輸或儲存所需之位元，同時又可正確保留資料源之原貌。而近年來由於受限於頻寬資源日益缺乏與儲存設備容量需求日增，因此，無失真資料壓縮技術已被廣泛應用在許多儲存裝置，如磁帶、硬碟、檔案伺服器與可移除式之快閃記憶體及 USB 裝置等，用來提高儲存裝置的容量。另外，於通訊網路應用方面，如區域網路、廣域網路、無線網路等，亦利用無失真之資料壓縮來達到資料傳輸效率提升之目的。

無失真壓縮技術最常被使用的有 LZ77[1] 、Huffman Coding[2]、Run Length Encoding[3]、Arithmetic Coding[4]、LZW[5]和 X-Match Pro[6]等，其中 LZ77 在 1977 年被 Abraham Lempel 和 Jacob Ziv 所提出來而且是最被廣泛使用的無失真壓縮技術之一。該演算法的概念十分簡單而且最主要的優點在於資料內容是未知或沒有經過統計的資料也可以得到不錯的壓縮效果。

原本的 LZ77 的資料結構有一個搜尋緩衝區(search buffer)和一個前看緩衝區(look-ahead buffer)，一開始搜尋緩衝區的內容預設為空的，而待編碼資料會被讀入前看緩衝區中，然後再以前看緩衝區的資料當作 Pattern，在搜尋緩衝區中去找到最長匹配的 Pattern，接著就產生編碼格式。LZ77 的演算法是以貪婪演算法的方式來得到最長的匹配字串，所以當搜尋緩衝區的容量越大，搜尋所花費的時間也相對增加，相對的，有匹配到的機率也會提高，可以增加壓縮率。因此選擇一個適當的搜尋緩衝區長度也是很重要的議題。但是主要影響壓縮率的原因來自於編碼格式的適當與否。以原來 LZ77 的編碼格式非常容易造成位元膨脹的問題。所以在本篇論文後面會深入去探討並提出修改之方法，最後再以軟體模擬得到壓縮數據。

2. LZ77 與 LZSS 演算法

2.1 LZ77 編碼演算法

步驟一：移動一個指向搜尋緩衝區內資料的指標(pointer)，一直到該指標所指的符號等於前看緩衝區內的第一個符號才停止移動。檢查指標所指符號後面的那個符號，看它的內容是否等於前看緩衝區內的第二個符號。如果相等，再往後看一個符號，重複比對，一直到不

相等為止。那些相等符號的總個數，稱為子序列匹配長度(match length)或符號串匹配長度。

步驟二： 重複步驟一，找出搜尋緩衝區內存在的所有匹配子序列，並決定出最長的匹配子序列才停止。

步驟三： 送出三個欄位的編碼結果[position, match length, next symbol]，其中position 代表從前看緩衝區到指標位置的距離(指標指到最長匹配子序列中的第一個符號)，匹配長度代表最長匹配子序列的長度(也就是符號相等的個數)，next symbol 代表緊接在最長匹配子序列後面第一個不匹配的編碼符號。

步驟四：重複步驟一到步驟三直到待編號資料都處理完畢才停止。

解碼演算法則依照和編碼演算法相反的動作進行。

為了更清楚的解釋 LZ77 編演算法，舉例說明，假設有一待編碼序列為：{ XYYCBXYYGCBXYCBGCBX }，開始時，第一個要編碼的符號是” X” ，因為搜尋緩衝區(存放已編碼序列)還是空的，所以找不到任何匹配符號，只能編碼送出(0,0,X);第二個欄位是 0 表示匹配長度為 0 即是完全無匹配，X 代表此待編碼符號。然後將已編碼的 X 從前看緩衝區移入搜尋緩衝區。步驟一結果如圖 1 所示：

編碼	搜尋緩衝區									前看緩衝區								輸 入 序 列	編碼結果
前										X	Y	Y	C	B	X	Y	Y	GCBXYCBGCBX	**[0,0,X]**
後									X	Y	Y	C	B	X	Y	Y	G	CBXYCBGCBX	

圖 1. LZ77 編碼步驟一

接下來，下一個要編碼的符號為”Y”，在搜尋緩衝區沒有找到相同的符號”Y”，所以匹配長度 0，故編碼成(0,0,Y)，如圖 2 所示：

編碼	搜尋緩衝區									前看緩衝區								輸 入 序 列	編碼結果
前									X	Y	Y	C	B	X	Y	Y	G	CBXYCBGCBX	**[0,0,Y]**
後								X	Y	Y	C	B	X	Y	Y	G	C	BXYCBGCBX	

圖 2. LZ77 編碼步驟二

接下來，下一個要編碼的符號為”Y”，在搜尋緩衝區找到 Y，指標距離前看緩衝區只有一個符號字元，(position=0)，匹配長度為 1，故編碼成(0,1,C)。然後將已編的 YC 從前看緩衝區移入搜尋緩衝區中，結果如圖 3 所示：

編碼	搜尋緩衝區									前看緩衝區								輸 入 序 列	編碼結果
前								X	Y	Y	C	B	X	Y	Y	G	C	BXYCBGCBX	**[0,1,C]**
後						X	Y	Y	C	B	X	Y	Y	G	C	B	X	YCBGCBX	

圖 3. LZ77 編碼步驟三

重複這些動作，經過八個步驟後，整個序列編碼完成，全部編碼結果如圖 4 所示：

步驟	搜尋緩衝區									前看緩衝區								輸 入 序 列	編碼結果
一										X	Y	Y	C	B	X	Y	Y	GCBXYCBGCBX	**[0,0,X]**
二									X	Y	Y	C	B	X	Y	Y	G	CBXYCBGCBX	**[0,0,Y]**
三								X	Y	Y	C	B	X	Y	Y	G	C	BXYCBGCBX	**[0,1,C]**
四						X	Y	Y	C	B	X	Y	Y	G	C	B	X	YCBGCBX	**[0,0,B]**
五					X	Y	Y	C	B	X	Y	Y	G	C	B	X	Y	CBGCBX	**[4,3,G]**
六	X	Y	Y	C	B	X	Y	Y	G	C	B	X	Y	C	B	G	C	BX	**[6,4,C]**
七	X	Y	Y	G	C	B	X	Y	C	B	G	C	B	X					**[3,1,G]**
八	Y	G	C	B	X	Y	C	B	G	C	B	X							**[6,3,”E”]**

圖 4. LZ77 編碼結果

編碼後的結果為：{(0,0,X),(0,0,Y),(0,1,C),(0,0,B),(4,3,G),(6,4,C),(3,1,G),(6,3,”E”)} ， 其中”E”代表檔案結尾。解碼則使用和編碼相反的過程。一開始，先讀入(0,0,X)，因為最長匹配為 0，表示符號 X 並不存在搜尋緩衝區中，直接將 X 解碼出來後，將 X 結果送出並將它移入搜尋緩衝區以供下個解碼搜尋，經過八個步驟後，整個序列解碼完成，解碼結果如圖 5 所示：

步驟	輸入	解碼結果	搜尋緩衝區									前看緩衝區							
一	[0,0,X]	X																	
二	[0,0,Y]	Y									X								
三	[0,1,C]	YC								X	Y								
四	[0,0,B]	B						X	Y	Y	C								
五	[4,3,G]	XYYG					X	Y	Y	C	B								
六	[6,4,C]	CBXYC	X	Y	Y	C	B	X	Y	Y	G								
七	[3,1,G]	BG	X	Y	Y	G	C	B	X	Y	C								
八	[6,3,”E”]	CBX	Y	G	C	B	X	Y	C	B	G								

圖 5. LZ77 解碼結果

當 LZ77 在完全沒有任何匹配的時候，還是會編碼來表示一個字元，這種情況下會發生位元膨脹（bit-expanding）的問題，導致壓縮率下降，而詳細情況在第三節會加以描述。接下來要介紹由 James Storer 與 Thomas Szymanski 在 1982 年針對 LZ77 缺點所提出的演算法 LZSS。

2.2 LZSS 編碼演算法

LZSS 演算法係針對LZ77演算法中沒有匹配到任何字元又產生3個欄位編碼結果，降低壓縮效果的缺點所提出的改進演算法。其方法與 LZ77 大致相同，僅在編碼欄位上有所不同。當有匹配時，產生的編碼結果為[flag, offset, match length]，其中 flag 被設為 1，offset 為從前看緩衝區到指標位置的距離，match length 為最長匹配子序列的長度；如果沒有匹配時，其所產生的編碼結果僅有 2 個欄位，格式為[flag, char]，其中 flag 被設為 0，char 代表這個要編碼的輸入符號。如此可以降低位元膨脹的程度，但因為在有匹配時，並沒有傳第一個不匹配的編碼符號，因此整個編碼完成的步驟數比 LZ77 的還要多。

使用剛剛 LZ77 的範例，經由十個步驟，可以將整個序列編碼完成，編碼結果如圖 6 所示

步驟	搜尋緩衝區									前看緩衝區								輸入序列	編碼結果
一										X	Y	Y	C	B	X	Y	Y	GCBXYCBGCBX	**[0,X]**
二									X	Y	Y	C	B	X	Y	Y	G	CBXYCBGCBX	**[0,Y]**
三								X	Y	Y	C	B	X	Y	Y	G	C	BXYCBGCBX	**[1,0,1]**
四							X	Y	Y	C	B	X	Y	Y	G	C	B	XYCBGCBX	**[0,C]**
五						X	Y	Y	C	B	X	Y	Y	G	C	B	X	YCBGCBX	**[0,B]**
六					X	Y	Y	C	B	X	Y	Y	G	C	B	X	Y	CBGCBX	**[1,4,3]**
七		X	Y	Y	C	B	X	Y	Y	G	C	B	X	Y	C	B	G	CBX	**[0,G]**
八	X	Y	Y	C	B	X	Y	Y	G	C	B	X	Y	C	B	G	C	BX	**[1,5,4]**
九	B	X	Y	Y	G	C	B	X	Y	C	B	G	C	B	X				**[1,3,2]**
十	Y	Y	G	C	B	X	Y	C	B	G	C	B	X						**[1,6,4]**

圖 6. LZSS 編碼結果

解碼則是使用和編碼相反的過程。如圖 7 LZSS 解碼結果所示：

步驟	輸入	解碼結果	搜尋緩衝區									前看緩衝區							
一	[0,X]	**X**																	
二	[0,Y]	**Y**									X								
三	[1,0,1]	**Y**								X	Y								
四	[0,C]	**C**							X	Y	Y								
五	[0,B]	**B**						X	Y	Y	C								
六	[1,4,3]	**XYY**					X	Y	Y	C	B								
七	[0,G]	**G**		X	Y	Y	C	B	X	Y	Y								
八	[1,5,4]	**CBXY**	X	Y	Y	C	B	X	Y	Y	G								
九	[1,3,2]	**CB**	B	X	Y	Y	G	C	B	X	Y								
十	[1,6,4]	**GCBX**	Y	Y	G	C	B	X	Y	C	B								

圖 7. LZSS 解碼結果

3. 位元膨脹問題

位元膨脹問題在第二節已經有提到其中一種，而另一種就是搜尋緩衝區的容量太小讓資料重複的部分出現在搜尋緩衝區裡的機率變小。但這兩種問題皆會造成壓縮之後的檔案變大而不是變小。解決的方法就是去增加搜尋緩衝區的容量，使匹配的機率變高，或是修改編碼格式來提昇壓縮的效果。

如果選擇增加搜尋緩衝區的容量去提昇壓縮率，會導致整個壓縮的時間隨著搜尋緩衝區的容量越大也跟著變長，而且製作硬體所需要的記憶體也是成本也會增加，所以一般不建議去增加容量來換取增加壓縮的效果，而是使用修改編碼格式的方法來提昇壓縮率。

在本篇論文中假設搜尋緩衝區跟前看緩衝區的大小各為 64 位元組與 32 位元組，所以 position、match length 和 next character 可以用 6 位元(0~63)、5 位元(0~31)以及 8 位元來表示，而每一個編碼格式的大小各為 19 位元。然後來看 LZ77 的演算法儘管搜尋緩衝區裡面沒有匹配到任何字串的時候，LZ77 還是會產生編碼格式(19 位元)來表示一個字元(8 位元組)，壓縮後卻比原始編碼資料多出了 11 個位元，此即為位元膨脹發生的主因。接下來利用以下六張圖來說明位元膨脹的問題。

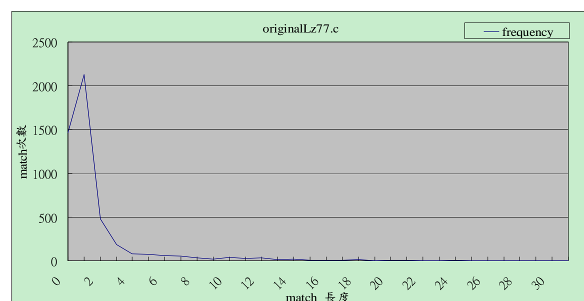

圖 9. originalLz77.c

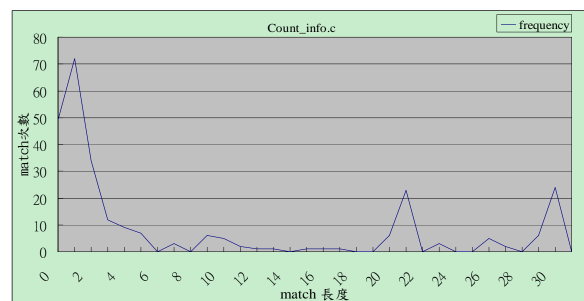

圖 10. Count_Info.txt

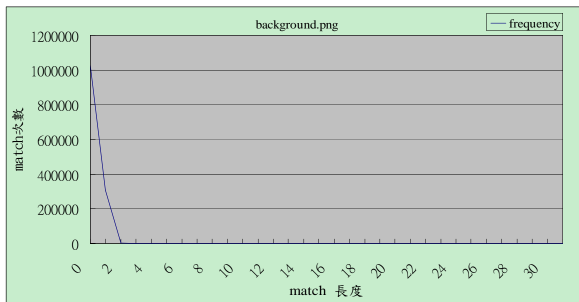

圖 11. background.png

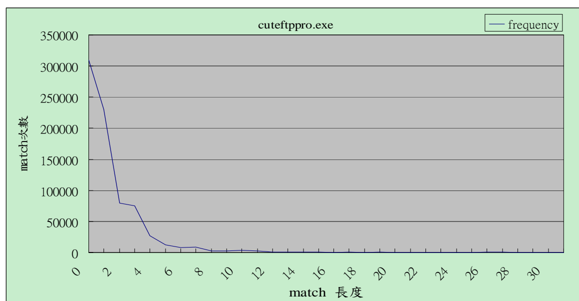

圖 12. cuteftppro.exe

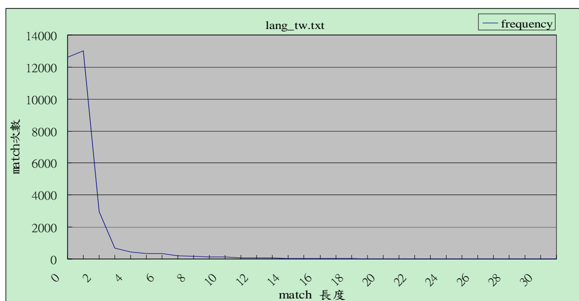

圖 13. lang_tw.txt

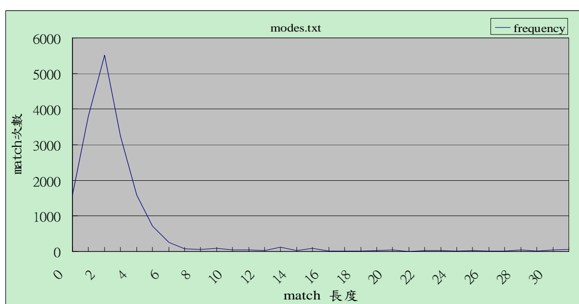

圖 14. modes.txt

在圖 9 到圖 14 分別是使用 C 語言實現 LZ77 演算法所模擬出來的匹配長度及次數的數據，因 LZSS 的圖表數據跟 LZ77 極為相似，所以不再呈現在本論文中。X 軸代表的是匹配長度，Y 軸代表的是該匹配長度的出現次數，而 LZ77 及 LZSS 的壓縮率顯示在表 1。先看表中 LZ77 壓縮率為正值的檔案，匹配長度是比較平均分散或沒有集中於 0 及 1，也就是用位元較少的編碼格式去表示了資料重複較長的部分，故壓縮率較佳。而壓縮率是負值的圖 11 和圖 13，其匹配長度全部集中於 0 及 1，而匹配長度越長者幾乎沒有出現，表示資料重複的部分變少也不能用較少的編碼格式去表示這些重複性低的資料，所以位元膨脹是相對非常嚴重，才會使得壓縮率呈現負值。本論文提出的修改方法將於下節中說明。

表 1. LZ77 與 LZSS 各別檔案的壓縮率

檔案名稱	原始大小（byte）	LZ77 壓縮率 %	LZSS 壓縮率 %
originalLz77.c	13,272	14.49	22.94
Count_Info.txt	2,489	73.95	75.76
background.png	1,655,573	-92.06	-20.91
cuteftppro.exe	1,966,080	6.80	21.29
lang_tw.txt	70,676	-5.00	9.75
modes.txt	81,497	47.32	48.88

4. Modified Method

對於匹配長度多為 0 跟 1 的資料而言，LZ77 及 LZSS 兩種演算法的壓縮效果皆不佳，為了改善其在這方面的缺點，本論文分別就兩種演算法所做的修改處予以說明。首先是對於 LZ77 編碼格式的改變：

a. 當匹配長度等於 0 時，輸出格式為：

[flag, next symbol]，flag 等於 0；

b. 當匹配長度大於 0 的時候，輸出格式為：

[flag, offset, match length, next symbol]，flag 等於 1；

此改變主要是增加一個旗標 flag 來表現是否有符號被匹配到，這樣對於匹配長度等於 0 的資料只需要用 9 個位元來表示，而不是 19 個位元。

接下來要介紹兩種修改 LZSS 的編碼格式的方法，首先瞭解原本 LZSS 的編碼格式的格式，在匹配長度大於 0 時，輸出為[flag, offset, match length]，flag 等於 1；等於 0 時輸出格式為[flag, next symbol]，flag 等於 0，offset 與匹配長度各別用 6 個位元以及 5 個位元去表示。

第一種方法針對匹配長度大於 0 的編碼格式去做改變，更改為[flag, offset, check flag, one bit]，當匹配長度等於 1 或 2 時，check flag 設為 0，one bit 等於 0 代表匹配長度為 1，等於 1 時代表匹配長度為 2。而匹配長度大於 2 輸出為[flag, offset, check flag, match length]，check flag 設為 1。

第二種方法在原本的 LZSS 的編碼格式中 match length 大於 0 的時候，利用靜態 Huffman Coding 來針對匹配長度出現的機率來做編碼，而匹配長度等於 0 的編碼格式沒有做任何改變。

5. 實驗結果

利用本篇論文修改的方法，搜尋緩衝區的大小為 64 位元組，前看緩衝區的大小為 32 位元組，並且以 C 語言所寫出來的程式來模擬取得壓縮率，如表 2 所示。

表 2. 修改 **LZ77** 後檔案的壓縮率

檔案名稱	LZ77 壓縮率 %	LZSS 壓縮率 %	修改 LZ77 壓縮率 %	修改 LZSS%	LZSS +Huffman %
originalLz77.c	14.49	22.94	29.18	33.61	36.84
Count_Info.txt	73.95	75.76	76.01	77.59	76.19
background.png	-92.06	-20.91	-12.30	-12.21	-9.33
cuteftppro.exe	6.80	21.29	26.51	28.48	32.17
lang_tw.txt	-5.00	9.75	17.63	21.83	25.38
modes.txt	47.32	48.88	48.39	54.95	58.74
平均壓縮率	7.50	26.28	30.90	34.04	36.67

從表 2 來看本論文所修改 LZ77 的方法可以將原本LZ77 平均壓縮率為7.5%提升到 30.9%，整整提昇了 23.4%。而且也比 LZSS 的壓縮率來的好。而修改 LZSS 與 LZSS 加上 Huffman coding 也可以將壓縮率提昇到一定的程度。雖然只有 background.png 壓縮率還是負值，但是原來 LZ77 就是針對文字來做壓縮，其對圖片以及 binary 的檔案壓縮率原本就不佳，除非再加上適應性 Huffman 編碼來針對個別檔案做處理，才可以對圖片跟 binary 的檔案做到更佳的壓縮效果，只是相對的硬體會變得更加複雜。

6. 結論

本論文針對 LZ77 與 LZSS 演算法中的缺點，分別以匹配長度出現的機率來修改編碼的格式，實驗數據顯示修改後的 LZ77 平均壓縮率由 7.5%提升到 30.9%，而修改後的 LZSS 也由 26.28%提昇到 34.04%。未來會利用 Systolic Array[9]的架構去實現 LZSS 的硬體架構以及搭配 ARM 的模擬板去實現軟硬體整合。

本論文研究經費部分由國科會計畫NSC98-2221-E-218-018 提供。

7. 參考文獻

[1] J. Ziv and A. Lempel, “A Universal Algorithm for Sequential Data Compression”, ***IEEE Trans. Inform. Theory***, *Vol. IT-23*, pp. 337-343, 1977.

[2] D. Huffman, ”A method for the construction of minimum redundancy codes.” ***Proc. IRE, 1958***, *Vol. 40*, pp. 1098-1101, Sep. 1952.

[3] S. Colomb, "Run Length Encoding," , ***IEEE Trans. Inform. Theory***, *Vol. IT-12*, pp 399-401, July 1966.

[4] G.G. Langdon Jr., "An Introduction to Arithmetic Coding," ***IBM J. Res. Development***, pp. 135-149, Mar. 1984.

[5] T. Welsh, "A Technique for high-performance data compression," ***IEEE Computer,*** *Vol. 17*, pp. 8-10, 1984.

[6] J. luis Nunez and Simon Jones, "Gbits/s Lossless Data Compression Hardware," ***IEEE Transactions on VLSI Systems***, *Vol. 11*, *No. 3*, June 2003.

[7] 陳培殷, “資料壓縮概論,” 滄海書局, 民 90 年

[8] R. Mehboob, S.A. Khan and Z. Ahmed, ”High Speed Lossless Data Compression Architecture,” ***2006. INMIC’06. IEEE***, pp.84-88, Dec. 2006.

[9] Abd El ghany, M.A., Salama, A.E, and Khalil, A.H., “Design and Implementation of FPGA-based Systolic Array for LZ Data Compression,” ***IEEE International Symposium on Circuits and Systems***, pp. 3691-3695, May 2007.

[10] Hu Yuanfu and Wu Xunsen, “ The Methods Of Improving The Compression Ratio Of LZ77 Family Data Compression Algorithms,“ ***3rd International Conference on Signal Processing,*** pp.698-701, Oct. 1996.