植基於符號組之改良式 BLIM 演算法

林柏豪 國立高雄大學 m0995510@mail.nuk.edu.tw
黃健峯 國立高雄大學 cfhuang15@nuk.edu.tw
陳建源 國立高雄大學 cychen07@nuk.edu.tw
林鈺峰 國立成功大學 aorborcord@gmail.com

***摘要*—本論文提出一套樣式比對演算法，以 BLIM 演算法為主，利用符號組(q-grams)的概念來加快比對速度，根據我們的分析，我們的方法在最佳情況下有 O($\frac{n}{m}$)的時間複雜度，其中 n 為序列長度，m 為樣式長度。根據實驗結果，顯示我們的方法與 BLIM 演算法比較，可減少 20 - 40%的比對時間。**

***關鍵詞*—樣式比對、BLIM 演算法、符號組**

***Abstract*—In this paper we present a new matching algorithm to improve the BLIM algorithm by using q-grams in the matching phase. Our analysis shows that, in the best case, the time complexity of our algorithm is O($\frac{n}{m}$), where n is the length of sequence, and m is the length of pattern. The experiments show that the matching time of our algorithm is reduced by about 20 – 40%.**

***keywords*—pattern matching、BLIM algorithm、q-grams**

一、前言

樣式比對，是一個相當傳統且應用性廣的研究主題，因此，在許多領域裡，很常見到樣式比對方法被應用的蹤跡，例如：影像處理[11][13]、資訊安全[3]、入侵偵測[12]、病毒掃描[6][8]、資料探勘[10][14][16]、生物資訊[9][20]等。樣式比對的主要問題，是針對在給定一個長度 n 的序列中，尋找長度 m 的樣式在序列中的所有位置。最直覺的解決方法為，對序列中的所有位置與符號都一個個比對，就可以輕易找尋出樣是存在於序列中的所有位置，可是這種方式卻需要花上 $n*m$ 次的比對次數，所以是相當耗時的。也因此，為了減少比對所花費的時間，許多學者提出各種演算法[2][4][7][15][18][19]來提升樣式比對的效能。我們將這些演算法整理以後，大致可分為四種類型：有限狀態機、符號關係、位元同步處理和過濾器。

有限狀態機之樣式比對主要在比對前根據樣式建立有限狀態機。有限狀態機將記錄符號之間的位置、排列、樣式字首(prefix)與字尾(suffix)等關係。在比對時，將讀入符號的比對過程轉變成狀態的改變，最著名的利用有限狀態機概念的演算法是在 1977 年由 Knuth、Morris 與 Pratt [4]所提出的 KMP 演算法。KMP 演算法只需要 n 次比對就能找出所有樣式的位置，但是建立有限狀態機需要大量的儲存空間。

符號關係的演算法，以 BM 演算法[19]為代表，由 Boyer 與 Moore 兩位學者所提出。BM 演算法在比對階段仍為符號比對，當符號比對不符合時，我們稱為比對失敗，此時必須查詢在比對前根據樣式關係所建立的「位移表」得到下個讀入符號的序列位置以加快比對速度，BM 演算法僅記錄樣式中符號之間的關係，不需大量的儲存空間。在最佳情況下，只需 n/m 次比對，但是最差情況仍需要 $n*m$ 次比對。

位元同步之樣式比對是由 Baeza-Yates 與 Gonnet [18] 利用位元同步處理狀態的轉換，提出 Shift-And 演算法(SA 演算法)，根據樣式中符號出現的位置以位元值記錄，形成狀態表，比對時查詢狀態表利用位元運算改變狀態。SA 演算法在任何情況下都需要 n 次比對，但是因為使用位元運算，所以提昇比對速度。之後 Navarro 與

Raffinot [7]，在位元同步處理的演算法利用移動視窗引入符號關係的概念，於 1998 年提出 BNDM 演算法。移動視窗將序列分割成一個個的子序列，在移動視窗範圍內比對時，執行位元同步處理之比對，當比對失敗或視窗內符號都比對過，就根據符號關係來移動到下一個視窗位置繼續比對，以省略不必要的比對步驟。而位元同步處理的演算法，無論是 SA 演算法或 BNDM 演算法，樣式長度受到計算機處理器能運算的位元長度(word size)限制問題，不利於長樣式之比對。到了 2008 年，Külekci [15]提出 BLIM 演算法解決此問題。將記錄符號出現在樣式中的位置改變為記錄樣式在移動視窗中的位置，使得樣式長度可以不受限制。BLIM 演算法也提出在移動視窗中的比對步驟的順序，使得比對步驟大幅減少，加快比對速度。

過濾器概念的演算法，以在 2007 年，Lecroq [17]所提出的比對演算法(Lecroq 演算法)最具代表。Lecroq 演算法每次讀入一組序列中連續的符號(q-grams)計算出赫序值，來判斷此符號組是否為樣式的一部分，如果可能為樣式，則將整段序列加以檢查，由於篩選可能造成誤判，增加不必要的比對次數，且篩選後的資訊沒有進一步有效利用，而形成浪費。因為篩選的過程極為快速，適用於長樣式的比對。根據 Lecroq 演算法的缺點，Ďurian、Holub、Peltola、Tarhio [2]四位學者在 2009 年提出 BNDMq 演算法，此為 BNDM 之改良，將過濾器概念應用在位元同步處理的演算法上，在比對階段時，移動視窗中的比對，先一次讀入一組符號來做篩選，當有可能為樣式時，才繼續讀入下一個符號直到比對成功或失敗，計算下一個移動視窗的位置繼續比對直到序列結束。BNDMq 演算法利用過濾器的概念，但在篩選後的資訊可進一步使用，因此，比 Lecroq 演算法更有效率。

本論文將架構在 BLIM 演算法，利用過濾器的概念，提出改良型樣式比對演算法，我們在每次移動視窗中的第一次比對時，讀入一組移動視窗中的數個符號，組成符號組(q-grams)，篩選樣式出現的可能，藉此減少比對次數，提昇演算法效率。

本論文組織架構如下，在第二章節，回顧 BLIM 演算法[15]。第三章節提出我們的演算法並舉例說明。演算法的分析與實驗結果，分別在第四章和第五章說明。最後第六章是我們的結論。

二、文獻回顧

樣式比對問題是在長度 n 的序列 $T[i]$，$0 \le i < n$，找到長度 m 的樣式 $P[j]$，$0 \le j < m$，在序列中的所有位置，其中序列 $T[i]$ 與樣式 $P[j]$ 中的符號都在大小為 σ 的字母系統 Σ 中。在本章節中，我們將回顧 BLIM 演算法[15]。

Külekci [15]在 2008 年，提出 BLIM 演算法解決位元同步處理概念的比對演算法受到計算機處理器所能處理的位元長度(word size)限制，樣式長度有其極限之問題。BLIM 演算法包含前處理階段與比對階段。前處理階段分為三個部分。首先根據樣式中符號在移動視窗中的位置建立查詢表 B；接著計算移動視窗移動的距離並記錄在位移表 S 中；最後，根據樣式長度決定在移動視窗中讀入符號的順序，建立成順序表 I。而比對階段，主要先建立一比對狀態值 D 來記錄比對過程，在移動視窗中根據順序表 I 先讀入第一個符號透過查詢表 B 儲存在比對狀態值 D，如果比對狀態值 D 不為 0，則依照順序表 I 讀入下一個位置的符號繼續更新比對狀態值，當移動視窗中的符號都讀入且比對狀態值 D 不為 0，則根據狀態值 1 的位置知道樣式出現在移動視窗的位置。若狀態值為 0，則讀入移動視窗後一個位置的序列符號查詢位移表 S，計算下一個移動視窗的位置，繼續比對直到序列結束。

1. 前處理階段

首先，定義樣式 P 的長度為 m 與計算機處

理器能處理的位元數(word size)為W，將樣式P排列成W列，而第k列為樣式P向右位移k個符號，$0 \le k < W$，如圖 1 為樣式"abcab" 和W為 8 所排列的結果，圖中，第 6 列，$k=6$，樣式向右位移 6 個符號，空白表示字母系統中的任意符號。涵蓋第 1 列到第 7 列的範圍為移動視窗，大小為ws，樣式P的長度為m，最大位移$W-1$個符號，所以 ws表示為

$$ws = W + m - 1 \text{。} \quad (1)$$

		ws										
		0	1	2	3	4	5	6	7	8	9	10	11
W	0	a	b	c	a	b							
	1		a	b	c	a	b						
	2			a	b	c	a	b					
	3				a	b	c	a	b				
	4					a	b	c	a	b			
	5						a	b	c	a	b		
	6							a	b	c	a	b	
	7								a	b	c	a	b

圖 1. 樣式"abcab"在移動視窗中的排列

接著建立查詢表B，針對字母系統中的符號ch，$ch \in \Sigma$與移動視窗中的位置po，$0 \le po < ws$，計算出其狀態值並儲存在查詢表B。狀態值為一W位元的值，第k位元記錄符號ch在移動視窗中第po個位置的第k列使否可能出現，如果不可能出現，則狀態值$B[ch][po]$的第k位元記為 0，其中$0 \le k < W$；反之，則記為 1。查詢表B中的狀態值$B[ch][po]$可表示成

$$B_k[ch][po] = \begin{cases} 0, & ch \neq P[po-k], \quad 0 \le po-k < m \\ 1, & otherwise \end{cases} \text{。} \quad (2)$$

以樣式"abcab"，字母系統Σ為 a、b、c、d 四個符號，$W=8$為例，所產生的查詢表B如表 1 所示，圖 1 中移動視窗的第 7 個位置，由第 8 列，即$k=7$，開始遞減向右排列，為" abcab□□□"，最左邊的符號"a"表示移動視窗的第 7 個位置的第七列，而"□"表示空白，此時符號"c"可能出現在第 0、第 1、第 2 及第 5 列，空白表示字母系統中的任意符號，所以可以為符號"c"，狀態值根據式(2)計算，為 00100111，每四個位元一組，記為 27。圖 2 為建立查詢表過程之虛擬碼。

表 1. 樣式"abcab"所產生的查詢表B

	0	1	2	3	4	5	6	7	8	9	10	11
a	FF	FE	FC	F9	F2	E5	BC	97	2F	5F	BF	7F
b	FE	FD	FA	F2	E9	D3	A7	4F	9F	3F	7F	FF
c	FE	FC	F9	F4	E4	C9	93	27	4F	9F	3F	7F
d	FE	FC	F8	F0	E0	C1	83	07	0F	1F	3F	7F

```
ComputeTableB(P, m, ws)
for i = 0 to ws - 1 do
    for all a ∈ Σ do
        B[a][i] = 1^W;
for j = 0 to W - 1 do
    tmp = 0^(W-i-1)10^i;
    for h = 0 to m - 1 do
        for all a ∈ Σ do
            B[a][i + j] &= ~tmp;
        B[P[j]][i + j] |= tmp;
```

圖 2. 建立查詢表B之虛擬碼

對於位移表S的建構，使用 Sunday[5]在 1990 年提出 Quick Search 演算法(QS 演算法)的「錯誤字元跳躍」的概念，在比對階段時，根據移動視窗的後一個之序列符號查詢位移表S得到位移距離，計算移動視窗的下個位置。位移表S記錄字母系統Σ中所有符號的位移距離，樣式

只需考慮在移動視窗中第 7 個樣式起始位置。根據字母系統 Σ 中的符號 c，$c \in \Sigma$ 出現在樣式中的位置，$P[j]=c$，$0 \le j < m$，位移值 $S[c]$ 設為 $ws-j$，若沒有出現則為 $ws+1$，表示成

$$S[c]=\begin{cases} ws-j, P[j]=c, 0 \le j < m \\ ws+1, \quad otherwise \end{cases}, c \in \Sigma, \quad (3)$$

表 2 為樣式"abcab"在字母系統 Σ 為 a、b、c、d，計算機處理器能處理的位元數(word size) W 為 8 的位移表 S，符號"d"沒有出現在樣式中，所以有最大的位移值 12。圖 3 為建立位移表 S 的虛擬碼。

表 2. 樣式"abcab"所產生的位移表 S

符號	位移值
a	8
b	7
c	9
d	12

```
ComputeTableS(P, m, ws)
for all a ∈ Σ do
    S[a] = ws + 1;
for j = 0 to m - 1 do
    S[P[j]] = ws - j;
```

圖 3. 建立位移表 S 的虛擬碼

```
ComputeTableI( m, ws)
i = 0;
for j = m – 1 down to 0 do
    k = j;
    while k < ws do
        I[i] = k;
        k = k + m;
        i = i + 1;
```

圖 4. 建立順序表 I 之虛擬碼

最後，計算移動視窗中讀入符號的順序，建立成順序表 I。為了讓比對更有效率，避免在移動視窗的最左邊與最右邊有太多的重複比對，一個簡單的方法就是在移動視窗中讀入符號的順序為 $m-i, 2m-i, \ldots, zm-i$，$zm-i < ws$，其中 $i=1,2,...,m$，以樣式"abcab"為例，比對順序為：4、9、3、8、2、7、1、6、11、0、5、10，圖 4 為計算順序表 I 之虛擬碼。

2. 比對階段

```
BLIM Algorithm ( P, m, T, n)
ws = W+ m – 1;
ComputeTableB(P, m, ws); //建立查詢表 B
ComputeTableS(P, m, ws); //建立位移表 S
ComputeTableI( m, ws); //建立順序表 I
i = 0;
while i < n do
    D = B[T[i + I[0]]][ I[0]] ;
    for j = 1 to ws-1 do
        D = D& B[T[i + I[j]]][ I[j]];
        if D = 0^W then
            break;
    if D ≠ 0^W then
        for h = W to ws-1 do
            if D&0^(W-j-1)10^j then
                OUTPUT(i + j);
    i = i + S[T[i + ws]];
```

圖 5. 比對階段之虛擬碼

進入比對階段，移動視窗與序列 $T[i]$，$0 \le i < n$ 的最左邊對齊，建立比對狀態值 D，比對狀態值 D 為一 W 位元的值，根據順序表 I 讀入第一個符號 $T[I[0]]$ 查詢查詢表 B 儲存在比對狀態值 D 中，

$$D = B[T[i + I[0]]], \quad (4)$$

接著依順序表 I 讀入符號查詢表 B，更新比對狀

態值D，表示為

$$D = D \& B[T[i + I[j]]], 0 < j < ws \text{ ，} \quad (5)$$

其中，符號"&"為 and 運算，若狀態值D為 0，表示比對失敗；若不為 0，繼續依照順序表I的順序讀入下一個序列符號更新狀態值D，直到發生比對失敗或整個移動視窗的符號都被讀入，當整個移動視窗的符號都被讀入且狀態值D不為 0 表示發現樣式，根據狀態值D中 1 的位置判斷樣式出現在移動視窗中的位置，接著根據移動視窗後一個位置的序列符號查詢位移表S，計算下一個移動視窗位置可以表示為

$$i = i + S[i + ws] \text{ ，} \quad (6)$$

繼續比對直到序列結束。

下頁中之表 3 為 BLIM 演算法在比對階段的執行範例，樣式P為"abcab"，$W = 8$，字母系統Σ={a, b, c, d}，序列T為"abcabcabdcabd"，長度為 13。依順序表I讀入符號直到移動視窗中的所有符號都讀入或是比對失敗，根據位移表移動到下一個視窗的位置。圖 5.為 BLIM 演算法在比對階段之虛擬碼。

三、我們的方法

我們的方法分為前處理階段與比對階段，前處理階段與 BLIM 演算法完全相同，分別建立查詢表B、位移表S和順序表I。主要的改進在於比對階段，同樣建立比對狀態值D，移動視窗與序列$T[i]$，$0 \le i < n$的最左邊對齊，在每次移動視窗中的比對一開始先讀入q個符號查詢查詢表B，儲存在比對狀態值D，表示為

$$D = B[T[i + I[0]]][I[0]] \& B[T[i + I[1]]][I[1]] \& \ldots \& B[T[i + I[q-1]]][I[q-1]] \text{ 。} \quad (7)$$

如果比對狀態值D不為 0，則依照順序表I讀入下一個位置的符號，繼續更新比對狀態值，即

$$D = D \& B[T[i + I[j]]], q < j < ws \text{ 。} \quad (8)$$

當移動視窗中的符號都讀入且狀態值D不為 0，則根據狀態值 1 的位置知道樣式出現在移動視窗的位置。若狀態值為 0，則讀入移動視窗後一個位置的序列符號查詢位移表S利用式(5)，計算下一個移動視窗的位置，繼續比對直到序列結束，我們利用 1 次讀入多個符號來篩選是否為樣式，接著再一個個讀入之後的符號與保留下篩選後的資訊作驗證，當樣式出現時，我們比 BLIM 演算法少了$q-1$次的比對次數。

```
My Algorithm ( P, m, T, n)
ws = W+ m – 1;
ComputeTableB(P, m, ws); //建立查詢表 B
ComputeTableS(P, m, ws); //建立位移表 S
ComputeTableI( m, ws); //建立順序表 I
i = 0;
while i < n do
    D = B[T[i + I[0]]][ I[0]] &…
        B[T[i + I[q-1]]][ I[q-1]];
    for j = q to ws-1 do
        if D = 0^W then
            break;
        D = D& B[T[i + I[j]]][ I[j]];
    if D ≠ 0^W then
        for h = W to ws-1 do
            if D&0^(W-j-1)10^j then
                OUTPUT(i + j);
    i = i + S[T[i + ws]];
```

圖 6. 比對階段之虛擬碼

表 3. BLIM 演算法在比對階段的執行

序列	*j*	*I*[*j*]	*B*[*ch*][*I*[*j*]]	*D*	備註
abcabcabdcabd	0	4	*B*[b][4] = E5 = 11101001	11101001	式(4)
abcabcabdcabd	1	9	*B*[c][9] = 9F = 10011111	10001001	式(5)
abcabcabdcabd	2	3	*B*[a][3] = F9 = 11111001	10001001	式(5)
abcabcabdcabd	3	8	*B*[d][8] = 0F = 00001111	00001001	式(5)
abcabcabdcabd	4	2	*B*[c][2] = F9 = 11111001	00001001	式(5)
abcabcabdcabd	5	7	*B*[b][7] = 4F = 01001111	00001001	式(5)
abcabcabdcabd	6	1	*B*[b][1] = FD = 11111101	00001001	式(5)
abcabcabdcabd	7	6	*B*[a][6] = BC = 10101101	00001001	式(5)
abcabcabdcabd	8	11	*B*[b][11] = FF = 11111111	00001001	式(5)
abcabcabdcabd	9	0	*B*[a][0] = FF = 11111111	00001001	式(5)
abcabcabdcabd	10	5	*B*[c][5] = C9 = 11001001	00001001	式(5)
abcabcabdcabd	11	10	*B*[a][10] = BF = 10111111	00001001	找到樣式在第 0 和第 3 個位置
abcabcabdcabd					查詢位移表 S[d] = 12，比對結束

表 4.我們的方法在比對階段的執行

序列	*j*	*I*[*j*]	*B*[*ch*][*I*[*j*]]	*D*	備註
abcabcabdcabd	0	4	*B*[b][4] = E5 = 11101001	00001001	式(7)，*q* = 4
	1	9	*B*[c][9] = 9F = 10011111		
	2	3	*B*[a][3] = F9 = 11111001		
	3	8	*B*[d][8] = 0F = 00001111		
abcabcabdcabd	4	2	*B*[c][2] = F9 = 11111001	00001001	式(8)
abcabcabdcabd	5	7	*B*[b][7] = 4F = 01001111	00001001	式(8)
abcabcabdcabd	6	1	*B*[b][1] = FD = 11111101	00001001	式(8)
abcabcabdcabd	7	6	*B*[a][6] = BC = 10101101	00001001	式(8)
abcabcabdcabd	8	11	*B*[b][11] = FF = 11111111	00001001	式(8)
abcabcabdcabd	9	0	*B*[a][0] = FF = 11111111	00001001	式(8)
abcabcabdcabd	10	5	*B*[c][5] = C9 = 11001001	00001001	式(8)
abcabcabdcabd	11	10	*B*[a][10] = BF = 10111111	00001001	找到樣式在第 0 和第 3 個位置
abcabcabdcabd					查詢位移表 S[d] = 12，比對結束

圖 6 為我們的方法的虛擬碼，而上一頁之表 4 為我們的方法在比對階段的執行範例，樣式 P 為"abcab"， $W=8$，q 設為 4，字母系統 Σ={a, b, c, d}，序列 T 為"abcabcabdcabd"，長度為 13。比對階段一開始依照順序表 I 先讀入 4 個符號，由式(7)計算得知比對狀態值 D 為 00001001，不為 0，繼續讀入下個符號，使用式(8)更新比對狀態值 D，直到比對失敗或移動視窗中的符號都被讀入。我們的方法在表 4 的範例中總共執行 9 次的比對，而表 3 的 BLIM 演算法則需要 12 次。

四、分析

在本章節中，我們除了提出我們的方法之複雜度外，也對符號組(q-grams) 的符號數量 q 與字母系統大小之關係進行分析。

針對我們的方法的複雜度分析，首先，分為前處理階段與比對階段。在前處理階段，我們建立了查詢表 B、位移表 S、順序表 I。由於我們在前處理階段與 BLIM 演算法完全相同，根據 Külekci [15]所提，計算查詢表 B 所需的時間複雜度為 $O((ws+Wm)\sigma)$，其中 σ 為字母系統 Σ 之大小，而空間複雜度為 $O(ws\sigma)$。位移表 S 其空間與時間複雜度分別為 $O(\sigma)$與 $O(\sigma+m)$。而順序表 I 的空間與時間複雜度皆為 $O(ws)$。接著，我們的方法在比對階段的時間複雜度最佳情況及最差情況為 $O(\frac{n}{m})$與 $O(\frac{nm}{W})$。對於平均時間複雜度部分，我們分為平均位移與平均符號比對數，所以平均時間可表示為

$$O(\text{平均比對次數}*\frac{n}{\text{平均位移}})。\quad (9)$$

假設符號呈均勻分布(uniformly distributed)的情形下，考慮所有位移的情況，我們的方法與 BLIM 演算法有相同的平均位移：

$$\frac{(\sigma-m)(W+m)+W+(W+1)+\ldots+(W+m+1)}{\sigma}$$
$$=W+m-\frac{m(m+1)}{2\sigma}\ ,\quad (10)$$

再來平均比對次數，考慮樣式 P 出現移動視窗中的機率 H 為

$$H=\frac{W\sigma^{ws-m}}{\sigma^{ws}}=W\sigma^{-m}\ ,\quad (11)$$

當樣式 P 出現，我們的方法在移動視窗中一開始先讀入 q 個符號，因此需要 $W+m-q$ 次比對。若樣式 P 沒有出現，在符號均勻分布下，平均比對次數為

$$\frac{(W-q+1)+(W+m-q)}{2}=W-q+\frac{m+1}{2}\ ,\quad (12)$$

所以可以得到整體的平均比對次數為

$$H*(W+m-q)+(1-H)*(W-q+\frac{m+1}{2})$$
$$=\frac{m-1}{2}(W\sigma^{-m}+1)+W-q+1\ ,\quad (13)$$

值得注意的是，當 $q=1$ 時，則為 BLIM 演算法。

由式(10)和式(13)可得到我們的方法之平均時間複雜度為

$$O(\frac{m-1}{2}(W\sigma^{-m}+1)+W-q+1*\frac{n}{W+m-\frac{m(m+1)}{2\sigma}})。\quad (14)$$

另外，我們針對符號組的符號數量q提出分析。直觀來說，符號組會因為符號排列組合的總數產生新的字母系統Σ'，而新符號是由數個符號所組成。假設，序列與樣式的符號為隨機分布，從序列讀入 1 個符號，與樣式中的特定位置符號相同的機率為$\frac{1}{\sigma}$，所以讀入q個符號比對成功的機率就為$\frac{1}{\sigma^q}$，讀入越多符號，也就是q值越大，$\frac{1}{\sigma^q}$越小，表示比對成功的機率越低，此可減少比對次數，增加比對效率。接著，考慮字母系統Σ之大小σ，σ越大也會使$\frac{1}{\sigma^q}$趨近於 0，圖 7 為針對字母系統大小σ為 4 及 20 時，各種q值的比對失敗機率$(1-\frac{1}{\sigma^q})$。可以看出，字母系統大小σ為 4，q值取 2 即能提昇比對失敗的機率，加快比對速度，而在q值為 4 與q值為 5 之間，比對失敗機率提昇幅度就沒那麼明顯，表示q值大於等於 4 時，可能不會有太大的效能提昇；字母系統大小σ為 20 則是在q值大於等於 2。

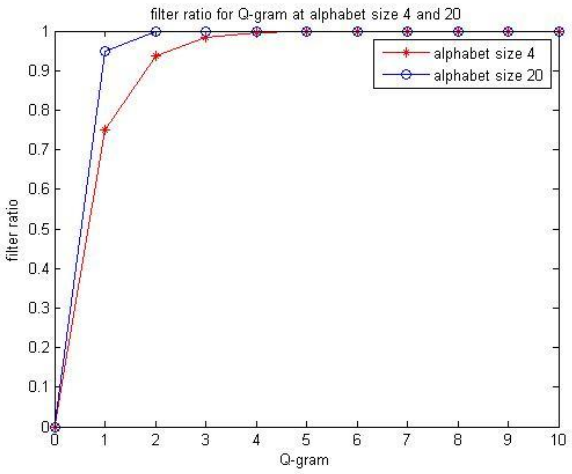

圖 7. 各種q值的比對失敗機率

五、實驗結果

本章節主要模擬我們的方法，並與其他四個演算法比較。我們使用的處理器為 Pentium4 3.0GHz、記憶體大小 2.5GB，Windows XP 作業系統之主機，在 Visual C++環境下執行我們的實驗。

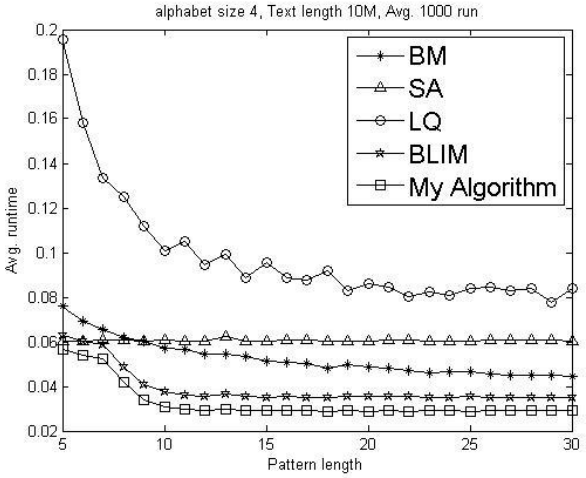

圖 8. 演算法在短樣式長度執行的結果

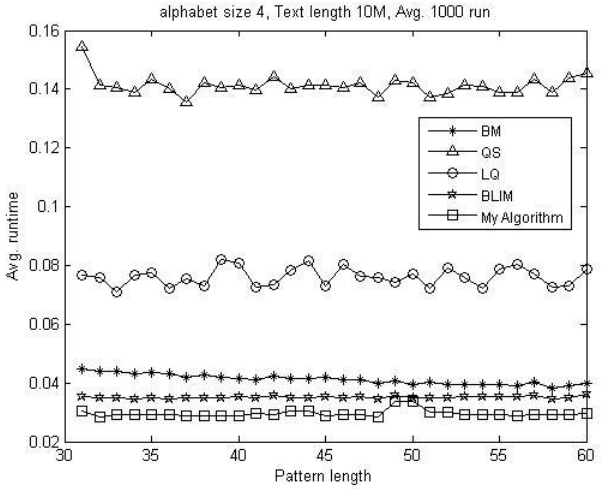

圖 9. 演算法在長樣式長度執行的結果

在第一個實驗中，字母系統大小為 4，隨機產生大小為 10MB 的序列，變動樣式長度，從長度 5 到 30，每種樣式長度執行 1000 次後取平均執行時間。我們的方法與 BM 演算法[19]、SA 演

表 5. 演算法在不同序列長度的執行時間

演算法	BM 演算法	QS 演算法	SA 演算法	LQ 演算法	BLIM 演算法	我們的方法
序列長度	平均執行時間					
100000	0.000532	0.001373	0.000439	0.001094	0.000282	**0.000311**
1000000	0.005242	0.015249	0.006101	0.009390	0.003860	**0.003120**
10000000	0.051185	0.152437	0.060959	0.090130	0.035208	**0.029140**
100000000	0.510665	1.529660	0.603417	0.913761	0.351612	**0.292099**

表 6. 演算法在不同字母系統大小的執行時間

演算法	BM 演算法	QS 演算法	SA 演算法	LQ 演算法	BLIM 演算法	我們的方法
序列長度	平均執行時間					
2	0.067392	0.383488	0.062648	0.154077	0.070177	**0.062764**
4	0.053189	0.159070	0.065271	0.093236	0.037226	**0.030833**
20	0.015169	0.044394	0.063165	0.062829	0.012958	**0.009852**
68	0.011129	0.032011	0.060639	0.050193	0.010765	**0.008677**
128	0.010791	0.031107	0.060721	0.064525	0.010445	**0.008339**

算法[18]、Lecroq 演算法[17](LQ 演算法)、BLIM 演算法[15]比較，結果如圖 8 所示，由圖 8 可知我們的方法在執行效率上優於其他四種演算法。

第二個實驗設定與第一個實驗相同，但樣式長度設為 30 到 60，由於 SA 演算法受到計算機可以記錄的位元個數限制無法在樣式長度大於 32 的情況下執行，所以我們替換為 QS 演算法[5]，其結果如圖 9，可以發現我們的方法在樣式長度較長的情況下，仍有較佳的執行效率。

接著第三個實驗，字母系統大小為 4，樣式長度為 16，隨機產生長度 100000、1000000、10000000、1000000000 之序列，每種序列長度皆執行 1000 次，取平均時間，比較我們的方法與 BM 演算法、QS 演算法、SA 演算法、Lecroq 演算法(LQ 演算法)、BLIM 演算法，其執行時間如表 5，在不同序列長度下，我們的方法的執行效率仍然較其他演算法佳。

第四個實驗則是考慮不同字母系統大小，字母系統大小為 2、4、20、64、128，樣式長度為 16，隨機產生長度 10000000 之序列，其各個演算法執行時間如表 6，即使在不同字母系統大小，我們的方法還是優於其他四種演算法。

表 7. 演算法之平均比對次數(一)

演算法	BLIM 演算法	BLIM 演算法 (無順序表)	我們的方法
樣式長度	平均比對次數（字母系統大小 2）		
10	6198.28	10835.7	**5077.97**
11	12054.70	21646.3	**9818.15**
12	17711.60	32606.8	**14336.40**
13	23065.60	43391.6	**18572.50**
14	28314.60	54137.4	**22706.40**
15	33565.00	64932.2	**26834.00**
16	38845.80	75764.8	**30999.10**

再來我們針對 BLIM 演算法和我們的方法，比較比對次數，在這個實驗，我們測試 BLIM 演

算法有使用順序表和BLIM演算法不使用順序表直接有左向右讀入符號，及我們的方法。隨機產生長度為100000，字母系統大小2、4、20的序列，樣式長度從10到16，每個情況執行1000次，記錄每次執行的比對次數，取平均。結果為表7、8、9，從結果可以觀察出，不使用順序表無論在字母系統大小、樣式長度都是需要最多的比對次數，使用順序表可以大幅減少比對次數，而我們的方法使用符號組(q-grams)的概念，使得比對次數更少。

表 8. 演算法之平均比對次數(二)

演算法	BLIM 演算法	BLIM 演算法 (無順序表)	我們的方法
樣式長度	平均比對次數（字母系統大小 4）		
10	2575.86	7986.6	**1633.05**
11	5067.62	15798.7	**3197.08**
12	7555.14	23668.4	**4749.61**
13	10053.40	31529.6	**6317.39**
14	12522.60	39276.5	**7867.41**
15	14942.70	46891.4	**9385.43**
16	17323.50	54404.2	**10874.40**

表 9. 演算法之平均比對次數(三)

演算法	BLIM 演算法	BLIM 演算法 (無順序表)	我們的方法
樣式長度	平均比對次數（字母系統大小 20）		
10	1314.27	5173.5	**673.25**
11	2581.13	10156.3	**1322.78**
12	3810.13	14981.4	**1953.99**
13	5001.25	19659.4	**2565.79**
14	6165.71	24231.4	**3164.31**
15	7300.26	28682.6	**3748.12**
16	8404.23	33007.9	**4316.98**

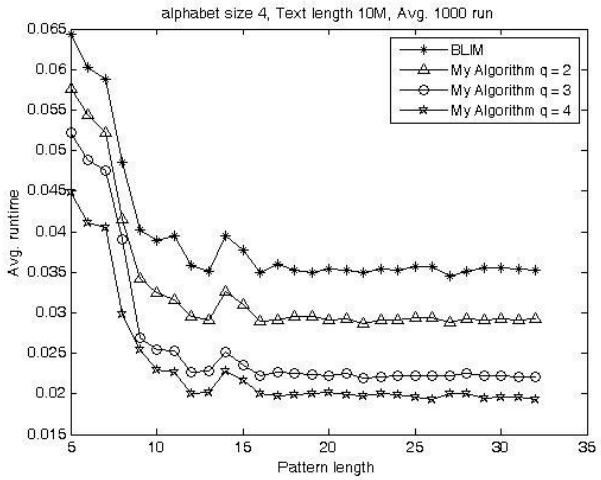

圖 10. 演算法在不同 q 值下執行的結果 (字母系統大小 4)

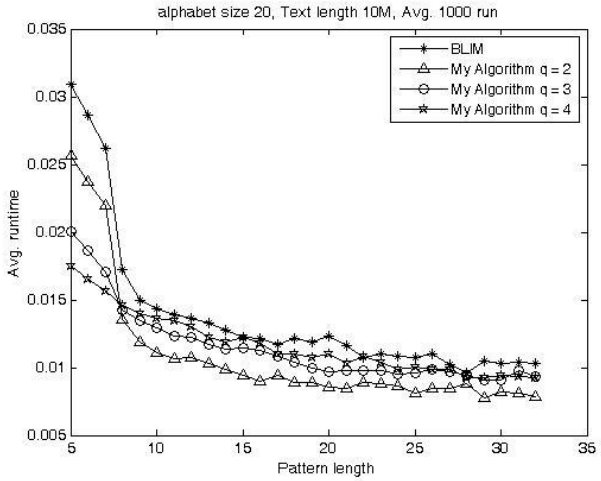

圖 11. 演算法在不同 q 值下執行的結果 (字母系統大小 20)

最後，根據我們的分析，演算法效率與符號組(q-grams)的 q 值和不同字母系統大小有關，在字母系統大小為 4 或 20，個別隨機產生長度 10000000 的序列，樣式長度從 5 到 32，q 值分別為 2、3、4，執行測試，值得注意的是 BLIM

演算法可視為q值為 1。實驗結果如圖 10、11 所呈現，可以觀察到，在字母系統大小為 4 的情況下，q值設為 4 的效率最佳，比 BLIM 演算法減少約 40%的比對時間；而字母系統大小為 20，則q值為 2 時最佳，約減少 20%的比對時間。

六、結論

本論文提出一套樣式比對演算法，以 BLIM 演算法為主，加入符號組(q-grams)的概念在比對階段，減少比對次數達到提昇比對效率，根據我們的分析，我們的方法在最佳情況下有 $O(\frac{n}{m})$ 的時間複雜度，其中n為序列長度，m為樣式長度。同時我們也針對符號組的大小與字母系統大小進行分析，在字母系統越大，符號組的大小則較小。更進一步地，我們從樣式長度、序列長度、字母系統大小來測試我們的方法之效能，由實驗結果可知，我們的方法明顯地優於其他四種演算法；同時，我們也測試符號組大小的改變對我們的方法的影響，與 BLIM 演算法比較，最多減少 40%的比對時間。

七、參考文獻

[1] A. V. Aho and M. J. Corasick, “Efficient string matching: an aid to bibliographic search,” Communications of the ACM, Vol. 18, Issue 6, pp. 333-340, June 1975.

[2] B. Ďurian, J. Holub, H. Peltola, and J. Tarhio, “Tuning BNDM with q-grams,” In: Proc. ALENEX 2009, the Tenth Workshop on Algorithm Engineering and Experiments, pp. 29-37, 2009.

[3] C. Haack and A. Jeffrey, “Pattern-matching spi-calculus,” Information and Computation, Vol. 204, Issue 8, pp. 1195-1263, August 2006.

[4] D. E. Knuth, J. H. Morris, and V. R. Pratt, “Fast pattern matching in strings, ” SIAM Journal on Computing, Vol. 6, Issue 1, pp. 323-350, 1977.

[5] D. M. Sunday, “A very fast substring search algorithm,” Communications of the ACM, Vol. 33, Issue 8, pp. 132-142, 1990.

[6] F. Yu, R. Katz, and T. V. Lakshman, “Gigabit rate packet pattern matching using TCAM,” In Proceedings of 12th IEEE International Conference on Network Protocols, pp. 174-183, October, 2004.

[7] G. Navarro, and M. Raffinot, “A bit-parallel approach to suffix automata: Fast extended string matching,” In Proceedings of Combinatorial Patern Matching 98, Springer-Verlag, pp. 14-33, 1998.

[8] G. Papadopoulos and D. Pnevmatikatos, “Hashing + memory = low cost, exact pattern matching,” In Proceedings of 15th International Conference on Field Programmable Logic and Applications, pp. 39-44, August 2005.

[9] J. Lin, Y. M. Zhu, and M. D. Lai, “Structural features of gr6 gene and its expression in colorectal neoplasm,” medical sciences, pp. 102-107, 2004.

[10] J. Zheng, T. J. Close, T. Jiang, and S. Lonardi, “Efficient selection of unique and popular oligos for large EST databases,” In Proceedings of 14th annual conference on Combinatorial pattern matching, pp. 384-401, June 2003.

[11] K. Zheng, X. Zhang, Z. Cai, Z. Wang, and B. Yang, “Scalable NIDS via Negative Pattern Matching and Exclusive Pattern Matcing,” In Proceedings of IEEE INFOCOM 2010, Main Conference, San Diego, CA, US, March, 2010.

[12] M. Alicherry, M. Muthuprasanna, and V. Kumar, “High speed pattern matching for network ids/ips,” In Proceedings of IEEE ICNP, pp.187-196, 2006.

[13] M. Alzina, W. Szpankowski, and A. Grama, “2D-pattern matching image and video compression: theory, algorithms, and experiments,” IEEE Trans on Image Process, Vol. 11, pp. 318-331, 2002.

[14] M. E. Califf, and R. J. Mooney, “Relational learning of pattern-match rules for information

extraction," In Proceedings of the 16th National Conference on AI, pp. 328-334, 1999.

[15] M. O. Külekci, "A method to overcome computer word size limitation in bit-parallel pattern matching," In Proceedings of ISAAC 2008: Proceedings of the 19th International Symposium on Algorithm and Computation, volume 5369 of Lecture Notes in Computer Science, pp. 496-506, 2008

[16] M. Wolverton, P. Berry, I. Harrison, J. Lowrance, D. Morley, A. Rodriguez, E. Ruspini, and J. Thomere, "LAW: A workbench for approximate pattern matching in relational data," In Proceedings of the Fifteenth Innovative Applications of Artificial Intelligence Conference (IAAI-03), 2003.

[17] T. Lecroq, "Fast exact string matching algorithms, " Information Processing Letters, Vol. 102, Issue 6, pp. 229-235, 2007.

[18] R. Baeza-Yates, and G. H. Gonnet, "A new approach to text searching," Communications of the ACM, Vol. 35, pp. 74-82, 1992.

[19] R. S. Boyer, J. S. Moore, "A fast string searching algorithm," Communications of the ACM, Vol. 20, pp. 762-772, 1977.

[20] Y. Huang, L. Ping, X. Pan, and G. Cai, "A fast exact pattern matching algorithm for biological sequences," In Proceedings of the International Conference on BioMedical Engineering and Informatics, Vol. 1, pp. 8-12, 2008.