影片以空間關係字串表示之壓縮

陳彥睿
淡江大學資訊管理學系碩士生
focke.chen@mail.im.tku.edu.tw

梁恩輝
淡江大學資訊管理學系暨研究所
ehliang@mail.im.tku.edu.tw

摘要

在影像資料庫中基於內容的相似影像尋取(Content-Based Image Retrieval，CBIR)是一個查詢影像的重要方法。物件之間的空間關係(spatial relation)是影像的重要特性之一。利用字串表示影像中物件之間的空間關係，以及利用空間關係進行空間推論及相似尋取之方法已被廣泛的討論。

影片中每一個畫面都可以視為一個靜態的影像，我們可以利用 2D B-string 表示每一個畫面中物件間的空間關係。在影片中前後的二個畫面往往差異都不大，為了有效減少重覆的字串，本論文利用字串比對的三種編輯運算表達相鄰的二個畫面 2D B-string 改變的部份，藉此縮短每個 frame 字串的長度，達到影片壓縮的效果，而且也能還原為原來的 2D B-string。

關鍵字：影片壓縮、字串比對、2D B-string

Abstract

In image database systems, Content-Based Image Retrieval (CBIR) is an important approach to image query. The spatial relationship between objects is one of the important features of the image. How to use strings to express the spatial relationship between objects and how to perform the inference and similarity retrieval have been widely discussed.

Because every frame in the video is a picture, we can use 2D B-string to describe the spatial relationship between objects in the frame. The difference between adjacent frames is not much. To reduce the repetition in the strings, we use the tree kinds of edit operations in string matching to record the difference between the 2D B-strings of adjacent frames. As a result, the length of the string can be reduced and the goal of compression can be reached. The original 2D B-string can be also recovered.

Keywords: video compression, string matching, 2D B-string

1. 前言

近年來，由於資訊技術的迅速發展，因而存在許多型態的多媒體資料，例如：靜態影像(image)、影片(video)。這些多媒體資料不論在商業的使用，或是個人的使用方面，都扮演著重要的角色。然而，若沒有一個合適的尋取方法的話，這些資料的可用性將會大大的降低。因此，多媒體資料的索引(index)及尋取(retrieve)技術變的愈來愈重要。

多媒體資料在尋取之前，必需有一套表示法，以對多媒體資料做索引，加速尋取的速度。影像資料及影片資料的索引技術可以大致歸納成兩類：低階的視覺特徵(low-level visual features)以及高階的關係特徵(high-level relationship features) [7]。低階視覺特徵包括影像中物件的顏色、紋理、形狀等；高階的關係特徵則包括影像中兩兩物件之間的相對位置、

距離、拓撲(topology)、空間(spatial)關係等。使用低階視覺特徵做為索引的例子包括QBIC[4]、VisualSEEK[13] 等系統，而 2D 字串[2]、2D C 字串[8]、[9]以及[12]則是使用高階關係特徵做為索引技術的例子。這兩種技術通稱為基於內容的相似尋取(Content-Based Retrieval，CBR)。在過去，基於內容的相似尋取被應用於影像相似尋取上[2]、[4]，此種技術則被稱為基於內容的影像相似尋取(Content-Based Image Retrieval，CBIR)。類似的概念則被延伸使用於影片資料庫的相似尋取問題上[4]。因此，我們可以將 CBIR 的技術，應用於影片相似尋取的問題上。

影片與影像是兩種非常相似的型態的資料。一部影片可以視為是由一串連續的靜態影像所組成的[3]，這些靜態影像稱為畫面(frame)。許多型態的資料都可以執行相似尋取，例如文字資料、影像資料、音訊(audio)資料及影片資料。影片資料的索引以及相似尋取的方法，可以從影像資料的索引及相似尋取技術延伸而來[1]。Shearer 及 Venkatesh 在[14]及[15]的研究中，將用於影像資料的索引表示法使用於影片資料的索引上。過去中也有以影像相似尋取的方法，而應用於影片相似尋取的問題上的也在過去的研究中被提出來了，例如：[11]以及[16]。

相似尋取是影像資料庫系統中重要的功能之一。在影像資料庫中，空間關係是被用來對影像資料做索引及相似尋取的方式之一。由Chang等人提出的2D字串是其中的一個例子。2D 字串是將影像中的物件分別投影至空間座標的 x、y 軸，以各種不同意涵的符號記錄投影資料，而用這些符號代表兩兩物件的空間關係。依照相同的概念，有許多改良 2D 字串的方法被提出，例如 2D C 字串、2D B-string[10]及 2D C^+字串[6]等等。而最大相似(maximal-likelihood)則是衡量兩張影像相似程度的標準之一[8]。在過去的研究中，[8]為採用最大相似的標準來衡量兩張影像相似度的例子之一。

既然影像可以視為由一連串的畫面(frame)所組成，每一個畫面又是一個靜態影像，我們可以用使用 2D B-string 應用於影片上使得每一個畫面都會有一組 2D B-string。一般來說，影片中前後二個 frame 的差異不會很大，因此前後二個畫面的 2D B-string 不會有很大的差異。因此如何壓縮每個畫面 2D B-string 的長度以提升傳遞速度為本論文重點。

在本論文中，我們利用 string match[18]來比對前後二個畫面的 2D B-string[]，將比對時所做的編輯運算(插入、刪除、改變)加以紀錄，在本論文中稱之為運算字串。因此假設一個影片中有 1~n 個畫面(frame<1> ~frame<N>)，只有 frame<1> 有完整的 2D B-string，而 frame<2>~frame<N>都是紀錄運算字串。因為運算字串只紀錄有改變的字串，不會重覆紀錄沒改變的字串，所以將可有效壓縮每個影片的長度。本論的架構如下。第二節中將使相關研究的探討。第三節將討論如何利用 string match 產生運算字串。第四節則是說明如何利用運算字串還原 2D B-string。第五節為演算法。最後第六節是本研究的結論。

2. 相關研究

2.1 空間關係表示法

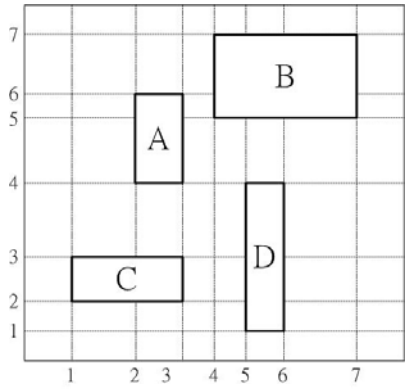
2D-strings是一種以物件為導向，根據物件之間的空間關係查詢影像之方法。一張影像經過影像處理(image process)和圖形辨識(pattern recognition)兩階段處理，並確認包含在影像中的所有物件後，影像可以被表示成符號圖(symbolic picture)，其中每一個符號代表原圖中的一個物件，且每一個物件皆以最小邊界矩形MBR(Minimum Bounding Rectangle，MBR)來代表整個物件。

在2D是一維字串的有序序對(ordered pair)，表示成(u, v)。它是將影像中的物件先以MBR做概略化的動作，再將MBR的重心分別對x軸及y軸做投影。其中第一個一維字串u表示物件投影至x軸上的空間關係，第二個一維字串v表示物件投影至y軸上的空間關係。2D-strings的結構表示為(S, R)，S與R的意義分別如下：

- S為符號圖中各個符號的集合。
- R為空間關係符號的集合，用來描述物件與物件之間的空間關係，包括”=”、”<”及”：”三種。“＝”代表兩物件投影到同一個位置上，“<”代表物件是左右或上下的空間關係，“:”代表兩物件在同一點上，即表示兩物件有重疊或覆蓋的情況。

2D C-string是一種延伸自2D-strings的概念，用來表示一張影像中物件間之空間關係的方法。對影像f而言，2D C_u-string(f)代表影像f投影在x軸上的2D C字串；2D C_v-string(f)代表影像f投影在y 軸上的2D C字串。在2D C string中，表示空間關係的符號集合為{“<”, “|”, “=”, “[“, “]”, “%”, “/”}。接著將兩影像相似度的比較轉變為兩組字串的比對。依據此觀念，陸續有學者提出改良的方法，如：2D B-string、2D C^+ string…等等。

因為在本論文中將利用 2D B-string 表達影像物件之空間關係，因此在此說明 2D B-string。2D B-string 表示成(u, v)，其中 u、v 分別表示圖中物件投影至 x-軸和 y-軸後，物件間之起始邊界和結束邊界所形成之字串。”＝”代表投影在同一點之邊界，例如圖一之 2D B_u-string 為 CAA=CBDDB，2D B_v-string 為 DCCD=ABAB)。



圖一、2D B-string 之範例

2D B-string 主要是利用物件的起始邊界與結束邊界的序值之比較進行空間推論。在 x-軸或 y-軸上一維字串 u 或 v 中，各個符號序值的定義為該符號在字串中的位置減去在此符號前出現的”=”個數，且計算該符號在字串的位置時，不包括”=”在內；換句話說，也就是影像中各個物件投影至 x-軸和 y-軸的起始邊界和結束邊界的排列順序。

投影之後的物件以兩個符號表示，分別為代表物件的起始邊界和結束邊界，因此一個物件在一維字串 u 和 v 中各有兩個序值，分別是起始的序值和結束的序值。例如在圖一中，令 begin(X)和 end(X)分別代表物件 X 的起始邊界之序值和結束邊界之序值，因此圖一的 u 字串各物件之序值如下：

begin(A)=2、end(A)=3

begin(B)=4、end(B)=7

begin(C)=1、end(C)=3

begin(D)=5、end(D)=6

v 字串各物件之序值如下：

begin(A)=4、end(A)=6

begin(B)=5、end(B)=7

begin(C)=2、end(C)=3

begin(D)=1、end(D)=4

2.2 字串比對

字串比對(Robert A. Wagner and Michael J. Fischer , 1974)是計算兩字串間，將一字串轉換成另一字串所需的最小成本(minimum cost)的編輯運算(editing operation)，字串比對的編輯運算有三種：

- 改變(change)：將一個字元改變成另一字元。例如字串 A 為 aook，字串 B 為 book，為了將字串 A 轉變為字串 B 將字串 A 的字元 a 改變為字元 b，所做的編輯運算稱之為改變。
- 刪除(delete)：將一個字串中刪除一個字元。例如字串 A 為 books，字串 B 為 book，將字串 A 字元 s 刪除使得字串 A 和字串 B 一致，此一編輯運算稱之為刪除。
- 插入(insert)：從一個給定的字串中插入一字元。例如字串 A 為 ook，字串 B 為 book，在字串 A 最前面插入字元 b，此一編輯運算稱之為插入。

為了說明字串比對，我們給予下列定義：

(1) A、B 是有限字元的字串

(2) A<i>是字串 A 中第 i 個字元，A<i:j>是字串 A 中第 i 個字元到第 j 個字元，也就是 A<i:j>=A<i> A<i+1> A<i+2>…A<j>。

(3) A(i)=A<1:i>表示字串 A 中第一個字元到第 i 個字元，也就是 A<1:i>=A<1> A<2> A<3>・・・A<i>。

(4) |A|是字串A 的字元數。

(5) D[i，j]=表示從字串A 的第1 個字元到第i 個字元轉變成字串B 的第1 個字元到第j 個字元所需的最小差距(distance)，且 0<=i<=|A|，0<=j<=|B|。

(6) γ(x->y)表示將x變成y所需的成本，其中 x=A<i>或是空的(Λ)，y=B<j>或是空的(Λ)，但是x、y 不能同時為空的(Λ)，且 0<=i<=|A|，0<=j<=|B|。若y為(Λ)表示將字元x刪除的成本；若x為(Λ)表示插入字元y所需的成本；如果x=y表示同字元，則 γ(x->y)=0。

我們將舉例說明字串比對運算之基本概念。假設字串A為adcbb，字串B為abcd，要將字串A轉換成字串B，假設字元比對的三個編輯運算：改變、刪除、插入的成本均為1。一開始表一根據字串A、字串B產生。表一中D[0,1]表示從空字串插入字元a需要成本為1，D[0,2]表示從空字串插入字元a及字元b需要2的成本。以此類推，D[0,4]表示從空字串插入a、b、c、d所需要的成本為4。D[1,0]表示為將字串A刪除字元a所需要的成本為1，D[2,0]表示將字串A刪除字元a、d需要的成本為2；以此類推，將字串A中a,d,c,b,b五個字元刪除的成本為5。接著將根據下列式子求得表一中灰色部份每一格的值：

D[i , j]=min(D[i-1 , j]+ γ (A<i> ->Λ) ,D[i , j-1]+ γ (Λ-> B<j>), D[i-1 , j-1]+ γ (A<i> -> B<j>))

D[i , j]可以根據D[i-1 , j]做刪除運算，或是D[i , j-1]做新增運算，或是D[i-1 , j-1]做改變運算求得，但是三條路徑各有不同的成本，而成本最小的將被選擇。以D[1,1]為例，三條路徑分別為D[0,1]做刪除運算，成本為2；第二條是從D[1,0]做新增運算，成本為2；最後一條是從D[0,0]做修改運算，成本為0，因為A<1>=B<1>都是字元a，所以不需要改變運算，所以此時的改變運算的成本用0計算，三個成本我們挑出最小的0為D[1,1]的值。如表二所示。若A<i>!=B<j>則要做改變運算，成本要用假設的1計算。再例如表三中的D[1,2]，A<1>!=B<2>。三條路徑分別為是從D[1,1]做新增運算，成本為1；第二條是從D[2,1]做刪除運算，成本為3；最後一條是從D[0,1]做改變運算，成本為2。三個成本挑最小的1為D[1,2]的值。如此就可以由左至右，由上至下依序推算出D[1,3],D[1,4], D[2,1]…D[5,4] 的值如表四所示。其中的D[5,4]的值就是將字串A(adcbb)轉換為字串B(abcd)所需要的最小成本3。為了方便說明本論文稱

表四這種紀錄所有D[i,j]值的表為編輯成本表。編輯成本表可由下例演算法求得。

➢ 編輯成本演算法：

Input：字串A、字串B
Output：編輯成本表D
Steps：

```
1   D[0,0]=0;
2   for i = 1 to |A| do
3      D[i,0]=D[i-1,0]+  γ(A<i> ->Λ);
4   for j = 1 to |B| do
5      D[0,j]=D[0,j-1]+  γ(Λ->B<j>);
6   for i = 1 to |A| do
7      for j = 1 to |B| do
8      { m1= D[i-1,j-1]+  γ(A<i> ->B<j>);
9        m2= D[i-1 , j]+ γ(A<i> ->Λ);
10       m3= D[i , j-1]+ γ(Λ-> B<j>);
11       D[ i ,j]=min(m1,m2,m3);
12     }
```

表一、編輯成本表初始化之範例

	j	0	1	2	3	4
i	D[i,j]	Λ	a	b	c	d
0	Λ	0	1	2	3	4
1	a	1				
2	d	2				
3	c	3				
4	b	4				
5	b	5				

表二、求的D[1,1]三個路徑之範例

	j	0	1	2	3	4
i	D[i,j]	Λ	a	b	c	d
0	Λ	0	1	2	3	4
1	a	1	0			
2	d	2				
3	c	3				
4	b	4				
5	b	5				

表三、求的D[1,2]三個路徑之範例

	j	0	1	2	3	4
i	D[i,j]	Λ	a	b	c	d
0	Λ	0	1	2	3	4
1	a	1	0	1		
2	d	2				
3	c	3				
4	b	4				
5	b	5				

表四、編輯成本表

	j	0	1	2	3	4
i	D[i,j]	Λ	a	b	c	d
0	Λ	0	1	2	3	4
1	a	1	0	1	2	3
2	d	2	1	1	2	2
3	c	3	2	2	1	2
4	b	4	3	2	2	2
5	b	5	4	3	3	3

完成編輯成本表後，我們就可以利用以下的演算法求得表中的最小成本路徑及將字串A轉變為字串B的最小成本編輯運算序列，令其為S。

➢ 最小成本編輯運算序列演算法：

Input：編輯成本表 D
Output：編輯運算序列 S
Steps：

```
1    i=|A|; j=|B|;k=1;
2    while ( i != 0 && j != 0 ) do
3    { if ( D[i , j]= D[i-1,j]+ γ(A<i> ->Λ))
4        { sk = delete( i );
5          i = i -1;
6       }else if (D[i , j-1]+ γ(Λ-> B<j>)) {
7          sk =insert(i ,B<j>);
8          j = j - 1;
9       }else if (A<i> != B<j>)
10       { sk =change( i , B<j>);
11         i = i – 1;     j = j - 1;
12       }
13     k=k+1
14   }
15   while ( i != 0 ) do
16   {   sk = delete( i );
17       i = i -1;
```

```
18        k=k+1;   }
19   while ( j != 0 )
20   {    sk = insert(i ,B<j>);
21        j = j - 1;
22        k=k+1;   }
```

$S=[s_1,s_2\cdots s_n]$，S 為 s_k 所成的序列，每一個 s_k 都代表一個編輯運算，使得字串 A 根據 S 包含的編輯運算順序處理後，可以使得字串 A 變為字串 B，其中每個 s_k 執行過後會產生一個 D[i,j]之值，令 s(i , j)表示產生 D[i , j]的編輯運算。根據求得 D[i , j]值的式子，可以得知 s(i , j)為何種運算，列述如下：

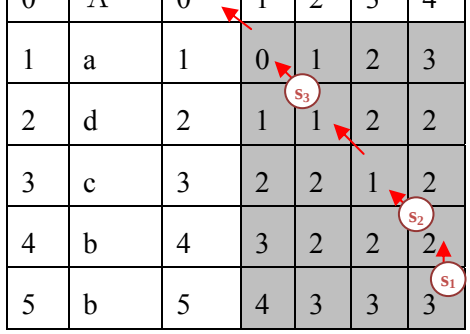
$$s(i,j)=\begin{cases}\text{delete}(i) & \text{if } D[i,j]=D[i-1,j]+\gamma(A<i>\to\Lambda)\\ \text{insert}(i,B<j>) & \text{if } D[i,j]=D[i,j-1]+\gamma(\Lambda\to B<j>)\\ \text{change}(i,B<j>) & \text{if } A<i>\neq B<j>\end{cases}$$

上述中 delete(i)表示刪除字串 A 第 i 個字元；insert(i , B<j>)表示將 B<j>插入字串 A 第 i 個字元的前面；change(i , B<j>)表示將字串 A<i>改變為 B<j>，但是要判斷是否 A<i> != B<j>，因為假設 A<i>==B<j>，則沒有必要做任何的編輯運算，只有 A<i> != B<j>才會做改變的編輯運算。演算法會從 D[|A|,|B|]開始，令 i=|A|、j=|B|，根據取得 D[i,j]的方式找到 D[i,j]在最小成本路徑中之前一個位置及 s(i,j)。例如在表五中 D[5,4]=D[4,4]+ γ(A<5> -> Λ)，所以 s(5,4)為 delete(5)，也就是 s_1=delete(5)，而且 D[5,4]前一位置為 D[4,4]，接著利用相同方法套用於 D[4,4]，如此重覆回溯運作直到 D[0,0]為止，即可求得最小編輯成本之路徑。並將所有 s(i , j)做依序加入 S。

根據演算法我們可以得到將字串 A(adcbb)轉變為字串 B(abcd)最小成本的路徑如表五中箭頭所示。其中 D[3,3]到 D[2,2]及 D[1,1]到 D[0,0]因為是相同字元的改變運算，因此事實上並沒有運算。所以 $S=[s_1,s_2,s_3]$，其中 s_1=Delete(5)、s_2=Change(4,d)、s_3=Change(2,b)。也就是說為了將字串 A 轉變成字串 B，我們將字串 A(adcbb)刪除第五個字元，將第四個字元改變為 d，最後將第二個字元改變為 b，即可得到 abcd 也就是字串 B。

表五、最小成本路徑表示

	j	0	1	2	3	4
i	D[i,j]	Λ	a	b	c	d
0	Λ	0	1	2	3	4
1	a	1	0	1	2	3
2	d	2	1	1	2	2
3	c	3	2	2	1	2
4	b	4	3	2	2	2
5	b	5	4	3	3	3



3. 運用編輯運算字串壓縮影像及還原

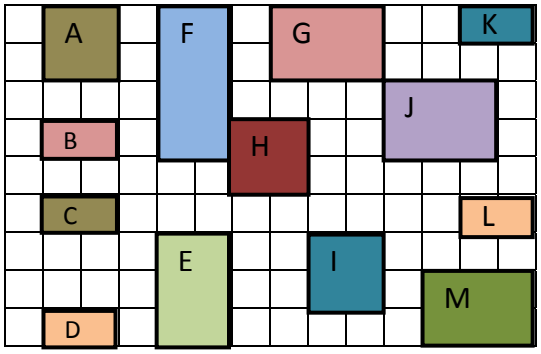
這節將會介紹如何使用上一節提到的字串比對演算法，比較影片前後二個畫面並產生運算字串。在影片中每一個畫面都可以視為一個靜態的影像，因此我們可以利用 2D B-string 表示每一個畫面。但是在影像中前後的二個畫面往往差異都不大，因此 2D B-string 會有很多重覆。為了有效減少重覆的字串，本論文利用字串比對的方法，計算出前後二個畫面 2D B-string 改變的部份，藉此縮短字串的長度。

字串比對運用到三種編輯運算(插入、刪除、改變)用來計算二字串間，將一字串轉換成另一字串所需的最小距離的編輯運算。本論文就是運用此方法到影片中，將影片中前後二個畫面的 2D B-string 做字串比對。並將字串比對所做的編輯運算用字串來表達並加以紀錄，取代影片中每一個畫面的 2D B-string 。為了用字串表達三種編輯運算，我們將編輯運算用字串表示如下：

- 插入(insert)：用符號"+"表示，在後面接要插入的位置及字串。例如："+2("表示在字串第二個位置上插入「(」。因此字串 ABBA 經過插入運算後變為 A(BBA。

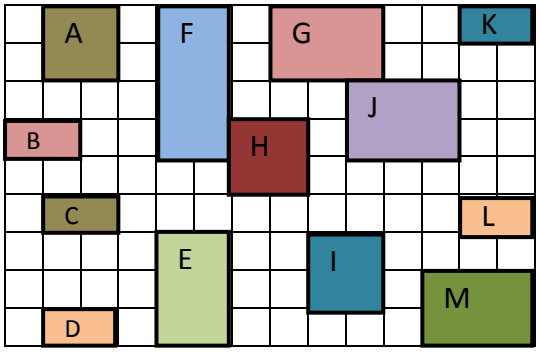
- 刪除(delete)：用符號”-“表示，後面接要刪除字元的位置。例如”-2”表示要刪除字串中位於第二的字元。因此字串 ABBA 經過刪除運算後變為 ABA。
- 改變(change)：用符號”~”表示，後面接被改變的字元位置及要改變的字元。如：”~2C”表示要將字串中第二個位置的字元改變為 C。字串 ABBA 經過改變運算後變為 ACBA

要將上一節介紹取得最短編輯路徑的方法運用於影片的 2D B-string 之前，我們要先將 2D B-string 的表示法做修正。在 2D B-string 中用”＝”代表二物件的邊界相同。本論文將”＝”以左括號”（”及右括號”）”取代，將邊界相同的物件字母放進左右括號內。如圖一之 2D B_u-string 表示為 CA(AC)BDDB 如此一來就方便將括號內的字元任意做排列，因為在 2D B-string 中括號內的字元不管如何排列其空間關係是不會改變的。但是在 string matching 中不同的排列順序都會影響最後編輯成本的值。為了將編輯成本的值不要受到因為括號內的排順序的影響。例如：字串 A 為 C(ABC)AB，字串 B 為 C(BAC)AB，將字串 A 變為字串 B 的成本為 D(|A|，|B|)=2。但是字串 A 及字串 B 所表現的空間關係是一樣的。為了避免上述的狀況，我們將二組 2D B-string 做 string matching 前要先將括號內的字元做排序，之後論文中範例的 2D B-string 都是將括號內字元做過排序的結果。再做字串比對以取得編輯運算字串。接下來將以圖二為一個含有四個畫面的影片舉例說明。由於 2D B v-string 的比對字串和 2D B u-string 是以相同方法求得，因此在圖二中只列出每個 frame 之 2D B u-string 並加以說明。



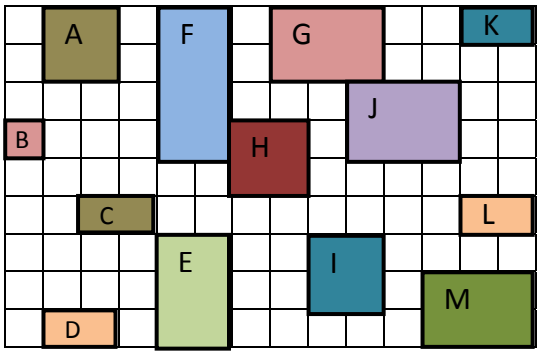
2D B_u-string：(ABCD)(ABCD)(EF)(EFH)G(HI)(GIJ)M(KL)J(KLM)

(a)Frame1



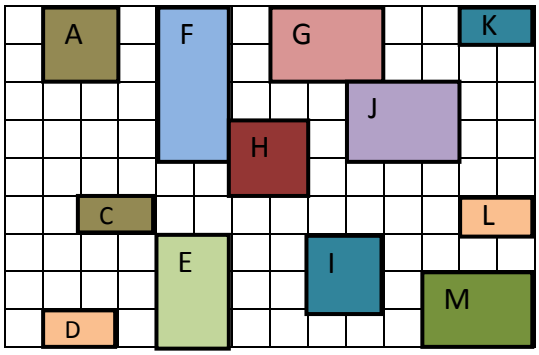
2D B_u-string：B(ACD)B(ACD)(EF)(EFH)G(HI)J(GI)M(JKL)(KLM)

(b)Frame2



2D B_u-string：B(ABD)C(AD)(CEF)(EFH)G(HI)J(GI)M(JKL)(KLM)

(c)Frame3



2D B_u-string：(AD)C(AD)(CEF)(EFH)G(HI)J(GI) M(JKL)(KLM)

(d)Frame4

圖二、影片範例

圖二影片中的四個畫面可以發現前後二個畫面物件的變化沒有很大，因此重覆的字元很多。為了減少重覆的字元達到壓縮影片的目地，我們運用上一介所介紹的字串比對產生運算字串來取代原本的 2D B-string。首先我們將 frame1 和 frame2 做字串比對並取得運算字串取代 frame2 的 2D B-string；再將 frame2 和 frame3 做字串比對取得運算字串取代 frame3 的 2D B-string，重覆相同動作直到 frameN-1 和 frameN 為止。圖二影片中四個畫面在 X 軸上之字串可以用運算字串表示如下：

frame1：

(ABCD)(ABCD)(EF)(EFH)G(HI)(GIJ)M(KL)J(KLM)

frame2：-37+34J-30+27J-9+7B-3+1B

frame3：+14C-10~7C~4B

frame4：-4-1

其中可以發現只有 frame1 維持 2D B-string 的表示法，之後的 frame 都以運算字串代替。例如 frame2 的字串中，+1B 表示在 frame1 的 2D B-string 第一個位置插入字元 B，-3 表示在 frame1 的 2D B-string 刪除第三個字元。藉由運算字串壓縮 frame2~frameN 的 2D B-string 以達到影片的壓縮。同時也可以利用 frame1 的 2D B-string 及 frame2 的編輯運算字串做還原。依照 frame2 的編輯運算字串來編輯 frame1 的 2D B-string 以還原 frame2 的 2D B-string，用相同方法依序還原 frame3~frameN 的 2D B-string 就可以還原整個影片。

4. 分析比較

我們將利用上節之範例說明本研究提出方法之壓縮效果。圖二影片中四個畫面在 X 軸上之字串若是沒有利用運算字串壓縮將分別表示如下：

frame1：

(ABCD)(ABCD)(EF)(EFH)G(HI)(GIJ)M(KL)J(KLM)

frame2：

B(ACD)B(ACD)(EF)(EFH)G(HI)J(GI)M(JKL)(KLM)

frame3：

B(ABD)C(AD)(CEF)(EFH)G(HI)J(GI)M(JKL)(KLM)

frame4：

(AD)C(AD)(CEF)(EFH)G(HI)J(GI)M(JKL)(KLM)

我們將四個 frame 所有字串長度分別為：frame1 為 42 字元長、frame2 為 42 字元長、frame3 為 42 長、frame4 為 40 字元長，總長度為 166 字元長。使用編輯運算字串壓縮後表示如下：

frame1：

(ABCD)(ABCD)(EF)(EFH)G(HI)(GIJ)M(KL)J(KLM)

frame2：-37+34J-30+27J-9+7B-3+1B

frame3：+14C-10~7C~4B

frame4：-4-1

四個 frame 的長度只有 frame1 維持 42 字元長，其它 frame2 為 24 字元長、frame3 為 13 字元長、frame4 為 4 字元長，總長度為 83 字元長。整個影片的壓縮比率為(166-83)/166=0.5。

利用編輯運算字串可以壓縮整個影片，壓縮比率的大小要看二個字串的差異越多，壓縮率越小；差異越少，壓縮率越高。而字串的差異決定於影片前後二個 frame 的物件變動，因此變動的物件越少，壓縮率越高。像是 frame3 與 frame4 的物件只有 B 改變，壓縮率相較其它 frame 之間的壓縮率高的許多。一般來說，影片中前後二個 frame 的物件變異都不大，因此利用編輯運算字串壓縮影片為一可行之方法。

5. 結論

在影片中每一個畫面都可以視為一個靜態的影像，因此我們可以利用 2D B-string 表示每一個畫面，以便做影像的索引及搜尋。但是在影像中前後的二個畫面往往差異都不大，因此 2D B-string 會有很多重覆。利用字串比對比較影

片中前後二個畫面，找到字串比對最短路徑將路徑中的編輯運算利用字串來表達，再利用求得的字串取代第二個畫面的 2D B-string。因為編輯運算字串只表達前後二個畫面不同的部份。因此當影片中前後二個畫面的物件多，且前後物件沒有太大的差異時，編輯運算字串將能會比原本的 2D B-string 字元數少的許多。一般來說，影片中前後二個 frame 的物件變異都不大，所以利用編輯運算字串達到壓縮影片之效果，而且也能還原為原來的 2D B-string。

參考文獻

[1] T. Arndt and S. K. Chang, “Image Sequence Compression by Iconic Indexing,” 1989 IEEE Workshop on Visual Languages, The Institute of Electrical and Electronic Engineers, IEEE Computer Society, Silverspring, MD, pp. 177-182, Oct. 1989.

[2] S. K. Chang, Q. Y. Shi and C. W. Yan, ”Iconic indexing by 2D-strings,” IEEE Trans. On Pattern Analysis and Matching Intelligence,PAMI-9,May,1987,pp.413-428.

[3] Sagarmay Deb, “Video Data Management and Information Retrieval,” Idea Group Inc. 2005.

[4] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Streele, and P. Yanker, “Query by Image and Video Content: The QBIC System,” Computer, vol. 28, no. 9, pp. 23-32, Sept. 1995.

[5] P.W. Huang and Y.R. Jean, “Spatial reasoning and similarity retrieval for image database system based on RS-strings,” Pattern Recognition, 1996,pp.2103-2114

[6] P.W. Huang and Y.R. Jean, “Using 2D C+-string as spatial knowledge representation for image database systems,” Pattern Recognition, 1994,pp.1249-1257.

[7] P. W. Huang and C. H. Lee, “Image Database Design Based on 9D-SPA Representation for Spatial Relations,” IEEE Trans. on Knowledge and Data Engineering, vol. 16, no. 12, 2004.

[8] F. J. Hsu and S. Y. Lee, “Spatial Reasoning and Similarity Retrieval of Images Using 2D C-String Knowledge Representation,” Pattern Recognition, vol. 25, no. 3, pp. 305-318, March 1992.

[9] F. J. Hsu and S.Y. Lee, “Similarity Retrieval by 2D C-Trees Matching In Image Database,” Journal of Visual Communication And Image Representation Vol.9, No. 1, March, 1998,pp.87-100.

[10] S. Y. Lee, M. C. Yang and J. W. Chen ,”2D B-string: a spatial knowledge representation for image database systems,” Proc.ICSC’92 Second Int. Computer Sci. Conf.,1992,pp.609-615.

[11] C. C. Liu and Arbee L. P. Chen, “3D-List: A Data Structure for Efficient Video Query Processing,” IEEE Trans. on Knowledge and Data Engineering, vol. 14, no. 1, 2002.

[12] M. Nabil, A.H.H. Ngu, and J. Shepherd, “Picture Similarity Retrieval Using the 2D Projection Interval Representation,” IEEE Trans. Knowledge and Data Eng., vol. 8, no. 4, pp. 533-539, Aug. 1996.

[13] J. R. Smith and S. F. Chang, “VisualSEEK: A Full Automated Content-Based Image Query System,” Proc. Fourth ACM Int’l Multimedia Conf., pp. 87-98, 1996.

[14] K. R. Shearer, S. Venkatesh, and D. Kieronska, “Spatial Indexing for Video Databases,” J. Visual Commun. Image

Representation, vol. 7, pp. 325-335, 1996.

[15] K. R. Shearer, D. Kieronska, and S. Venkatesh, “Resequencing Video Using Spatial Indexing,” J. Visual Languages Comput., vol. 8, pp. 193-214, 1997.

[16] K. R. Shearer, H. Bunke, and S. Venkatesh, “Video Indexing and Similarity Retrieval by Largest Common Subgraph Detection using Decision Trees,” Pattern Recognition, vol. 34, pp. 1075-1091, 2001.

[17] H. Tamura and N. Yokoya, “Image database systems: a survey,” Pattern Recognition, Vol. 17, NO. 1, 1984,pp.29-43.

[18] R. A. Wagner and M. J. Fischer, “The String-to-String Correction Problem,” Journal of the Association for computing Machinery, Vol. 21, No. 1, January 1974,pp.168-173.