基因序列比對的演算法

動態規劃

演算法就好像一套烹飪的食譜，
裡頭記載著烹飪名菜所需的材料及烹飪的細節與步驟，
任何人只要遵循這些細節和步驟，就可料理出這道名菜。

盧錦隆

隨著DNA定序技術的不斷提升以及各種基因體計畫的進行，越來越多的生物序列資料被建構成資料庫，並置於網際網路上免費供人使用。例如存放核苷酸序列資料的GenBank、EMBL-Bank和DDBJ等，以及存放蛋白質序列資料的Swiss-Prot、TrEMBL和PIR等。據估計，平均每15個月這些資料庫的內容就會增加一倍。利用這些資料，生物學家再藉由生物資訊和計算生物的分析工具，便可以了解許多生物問題。

在諸多生物資訊的分析工具中，序列比對是一個相當重要的研究工具，它可以比較及分析出序列之間的相似程度。相似度高的序列彼此間會有相似的結構及功能，這意味著它們可能源自共同的祖先。

在諸多生物資訊和計算生物的分析工具中，序列比對（sequence alignment）是一個基本且相當重要的研究工具，它可以比較及分析出兩條或多條序列之間的相似程度。相似度高的序列彼此間會有相似的結構及功能，這意味著它們可能源自共同的祖先。因此，生物學家一旦拿到未知功能的DNA或蛋白質序列時，最常做的事情就是利用序列比對工具搜尋資料庫，看看是否有已知註解功能的序列與手中未知功能的序列相似者，藉此推測手中序列的生物功能。

過去這種基因研究的工作，生物學家得純靠手工進行序列資料庫的比對和搜尋，通常得花費數年才能完成。現在利用電腦比對搜尋，可能只需幾秒鐘而已，眞是不可同日而語。

試想在電腦尚未普及的年代，生物學家首先得把整個資料庫的序列（幸好當時的資料量還不算大）印出並貼在牆上，接著把手中的序列寫在另一張紙上，然後一邊喝著咖啡，一邊用眼睛來回比對和搜尋手中及牆上的序列資料。這是件多麼費時又辛苦的工作啊！拜電腦之賜，生物學家不用再過這種苦日子了，現在只要動一動手指，就可完成序列資料庫的比對和搜尋。

當然空有電腦而無序列比對軟體的輔助也是行不通的！

http://www.vigyaancd.org/images/clustalx.jpg

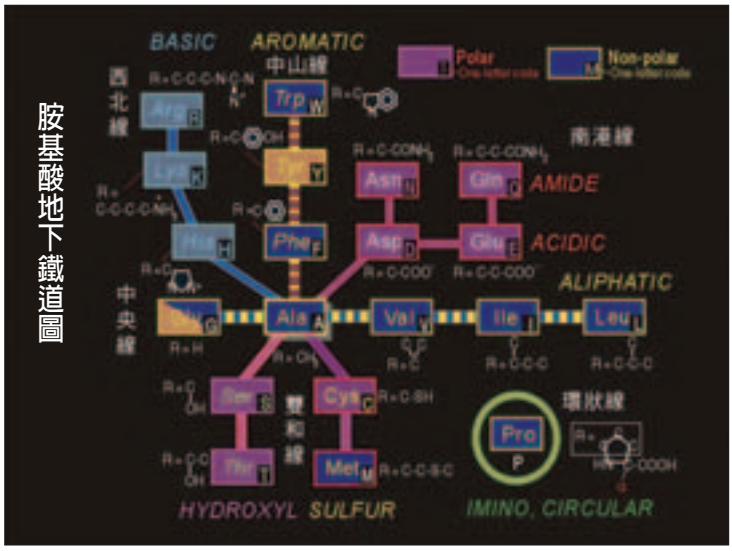
拜電腦之賜，生物學家只要動一動手指，就可完成序列資料庫的比對與搜尋。

但是要如何才能設計出一套有效率的序列比對工具呢？關鍵在於演算法。在計算機科學的領域中，演算法是一門基本且實用的學科。簡單來說，演算法就好像一套烹飪的食譜，裡頭記載著烹飪名菜（基因序列比對工具）所需的材料（輸入資料）及烹飪（比對）的細節與步驟，任何人（程式設計師）只要按照這些細節和步驟，就可料理（設計）出這道名菜（基因序列比對工具）。

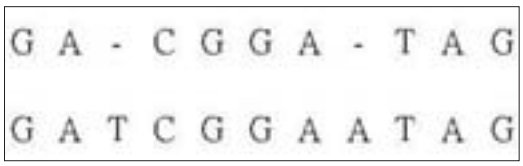
事實上，任何利用電腦從事研究的人或多或少都會用到演算法。通常我們會用執行時間複雜度來衡量一個演算法是否有效率。演算法的時間複雜度往往表示成一個與 n 有關的函數，n 是輸入資料的大小。若時間複雜度是一個多項式函數，例如 n^k，其中 k 是常數，那麼這個演算法就被稱爲有效率的演算法。反之若只能表示成非多項式函數，例如指數函數 k^n，其中 k 是常數，則稱爲沒有效率的演算法。

一般而言，我們會把DNA和蛋白質分別看成是由4和20個英文字母所組成的序列或字串，因爲他們分別是由4種核苷酸和20種胺基酸所組成的。對DNA而言，突變是非常平常的事情，也是自然的演化過程。藉由基因的突變，生物可以適應自然環境的改變。

常見的DNA突變有3種，分別是取代、插入及刪除。所謂取代，是指把一個字母（即核苷酸）用另外一個字母取代，而插入或刪除，是指在DNA序列的某一個位置插入或刪除一個新的或舊的字母。譬如我們把DNA序列AAGT的第1個字母A用T取代，然後把第3個字母G刪除，接著在序列的尾巴插入新的字母C，最後便可以得到一個新的DNA序列TATC。這整個突變的過程就如同大自然在對DNA序列進行編輯一般，所以上述3種突變又稱爲DNA的編輯運算。



胺基酸可依據其結構特徵來分類



兩條DNA序列的比對

通常生物學家會利用所謂的編輯距離，來衡量兩條DNA序列之間的相異程度。生命總是朝著最短路徑進行演化，所以兩條序列之間的編輯距離被定義爲：把其中一條序列編輯轉成另外一條序列，所需最少的編輯運算個數。兩條DNA序列之間的編輯距離越小，代表它們之間的相似程度越高。從演化的觀點來說，這意味著它們演化自同一個祖先（即所謂的同源），所以彼此間應該會有相似的結構及功能。

通常生物學家會比較喜歡利用比對來衡量兩條序列之間的相似程度。拿GACGGATAG和GATCGGAATAG這兩條DNA序列來說，乍看之下這兩條長度不同的DNA序列似乎不太相似。但是，當我們把它們重疊在一起，並在第1條序列的第2個和第3個字母之間與第6個和第7個字母之間分別插入一個空白字，就可發現其實這兩條DNA序列還蠻相像的。這種序列重疊的方式，就稱爲序列的比對。

我們可以在兩條序列的任意位置上插入一個或多個空白字，目的是讓相同或相似的字母能夠儘量對齊，但要特別注意的是不能讓兩個插入的空白字對齊在一起，因爲這樣對衡量序列之間的相似程度並無幫助。因此，字母之間

AGG-ACTA ACGTA-TA	A-G-GACTA ACGT-A-TA	AGGAC-TA-- ---ACGTATA

3種序列的對齊方式，若配對的欄位給1分，配錯、插入和刪除的欄位各給－1分，則圖中最左邊的對齊方式得2分，中間的對齊方式得1分，最右邊的對齊方式得－2分。

對齊的方式就只有2種：字母與字母的對齊，以及字母與空白字的對齊（即所謂的開gap）。當然，兩條序列之間的對齊方式不單單只有1種。例如對AGGACTA與ACGTATA這兩條DNA序列而言，至少就有3種對齊的方式。

事實上，每一種序列對齊方式都對應著一組編輯運算，可以把一條序列編輯轉成另外一條序列。以下舉例說明我們可以用3個編輯運算，把AGGACTA序列轉成ACGTATA序列。首先用一個取代編輯把第1個G轉成C（對應第2個欄位），然後再用一個插入編輯把T插在第3個與第4個字母之間（對應第4個欄位），最後再利用一個刪除編輯把C拿掉（對應第6個欄位）。正因爲如此，我們把第2個欄位稱爲配錯，第4個欄位稱爲插入，第6個欄位稱爲刪除，其他欄位稱爲配對。

李明提供

兩條DNA序列之間的編輯距離越小，代表它們之間的相似程度越高。從演化的觀點來說，這意味著它們演化自同一個祖先（即所謂的同源）。

一般而言，生物學家會根據某一種計分函數計算出每一種對齊方式的得分。就3種對齊方式而言，生物學家會認爲以上述的方式得分最高。計分函數是用來描述著兩個字母（包括空白字）之間的對齊分數，最簡單且最常用的計分方式，是把所有欄位的字母對齊分數加總起來做爲該對齊方式的得分。所謂的兩條序列比對問題，就是在給定的兩條序列與一種計分函數前提之下，找出一種得分最高的對齊方式。

如何設計出有效率的演算法來解決兩條序列比對的問題呢？當要對齊的序列長度還算小的情況下，我們可以採用暴力法或窮舉法來解決這個問題：就是列舉出所有的對齊方式，然後再從中找出得分最高的對齊方式。但是，若要對齊的序列長度很大，這種暴力或窮舉的方法就行不通了！因爲單單要列舉出所有的對齊方式，本身就是一件困難的事情。

假設要對齊的兩條序列長度都是n，那麼它們之間總共就有$\binom{2n}{n}\approx\frac{2^{2n}}{\sqrt{\pi n}}$個對齊方式。換句話說，對齊方式的個數會隨著序列的長度呈指數成長。

以人類整個基因體來說，序列的總長約爲30億，那麼兩條人類基因體序列的對齊總個數會是一個超級天文數字。即使利用全世界所有的電腦來做這件事情，我們（甚至我們的子孫）恐怕都無法在有生之年看到結果，這對電腦而言簡直是「不可能的任務」！幸好，我們可以利用動態規劃（dynamic programming）技巧，設計出一個簡單又有效率的演算法來解決兩條序列比對的問題。

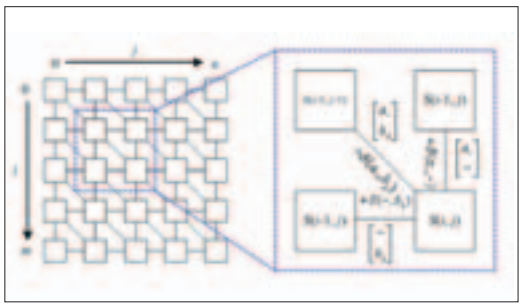
首先我們定義一些符號，以便能夠簡單及清楚地介紹兩條序列比對問題的動態規劃演算法。令$A=a_1a_2\cdots a_m$和$B=b_1b_2\cdots b_n$分別表示長度各爲m和n的兩條序列；令$Align(i,j)$表示兩條序列片段$A_i=a_1a_2\cdots a_i$和$B_j=b_1b_2\cdots b_j$之間

分數（或相似度）最高的對齊方式，其中 i 和 j 的範圍分別是 $1 \leqq i \leqq m$ 與 $1 \leqq j \leqq n$；令 $S(i,j)$ 表示 $Align(i,j)$ 的分數。我們的目的就是要求得兩條序列之間最佳的分數 $S(m,n)$ 及對齊方式 $Align(m,n)$。

在這之前，先把焦點放在 $S(i,j)$ 和 $Align(i,j)$ 的身上，因爲若有辦法對任意的 i 和 j 都能計算出 $S(i,j)$ 和 $Align(i,j)$，自然就可以求得 $S(m,n)$ 和 $Align(m,n)$。首先，我們把 $Align(i,j)$ 切成左右兩半：右半部只含 $Align(i,j)$ 的最後一個欄位，左半部則包含剩下的欄位。在這種情況之下，$S(i,j)$ 等於左半部的分數加上右半部的分數。不管 $Align(i,j)$ 的對齊方式爲何，最後一個欄位的對齊方式就只有 3 種：字母對齊字母（即 a_i 對齊 b_j）、字母對齊空白字（即 a_i 對齊 $-$）、以及空白字對齊字母（即 $-$ 對齊 b_j）。

若是第 1 種情形，那麼右半部的分數等於 a_i 對齊 b_j 的分數（用 $\delta(a_i, b_j)$ 表示），而左半部的分數等於 $A_{i-1}=a_1a_2\cdots a_{i-1}$ 與 $b_{j-1}=b_1b_2\cdots b_{j-1}$ 之間對齊的最高分數（即等於 $S(i-1,j-1)$）。換句話說，$S(i,j)=S(i-1,j-1)+\delta(a_i,b_j)$。同理可證，若是第 2 種情形，則 $S(i,j)=S(i-1,j)+\delta(a_i,-)$；第 3 種情形，則 $S(i,j)=S(i,j-1)+\delta(-,b_j)$，其中 $\delta(a_i,-)$ 和 $\delta(-,b_j)$ 分別表示「a_i 對齊空白字」與「空白字對齊 b_j」的分數。綜合這 3 種情形，我們可以把 $S(i,j)$ 表示成一個遞迴函式如下：

$$S(i,j)=max\begin{cases} S(i-1,j-1)+\delta(a_i,b_j), \\ S(i-1,j)+\delta(a_i,-), \\ S(i,j-1)+\delta(-,b_j) \end{cases}$$



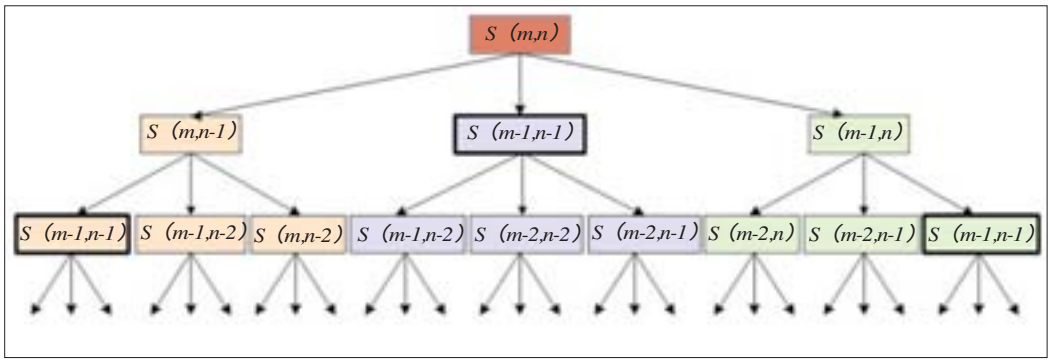
計算 $S(m,n)$ 的二維矩陣

這個遞迴函式關係就是動態規劃演算法最主要的特色之一，它告訴我們一個很重要的解題精神，就是把一個大的問題分解成幾個小的問題來解，一旦小問題都解決了，大問題就可以被輕易地解決了。

例如我們可以把計算 $S(m,n)$ 的問題分解成計算 $S(m-1,n-1)$，$S(m-1,n)$ 和 $S(m,n-1)$ 的 3 個小問題，然後再把這 3 個小問題的解代入遞迴函式，就可求出 $S(m,n)$。同樣的道理，可以把計算 $S(m-1,n-1)$ 的小問題，分解成計算 $S(m-2,n-2)$、$S(m-2,n-1)$ 及 $S(m-1,n-2)$ 這 3 個更小的問題。如此遞迴下去，我們可以把整個計算 $S(m,n)$ 過程用一顆樹狀圖來描述。

然而，這種由上而下計算 $S(m,n)$ 的方式會十分耗時，因爲整顆樹會有指數個節點，且每一個節點都需要花時間去計算其遞迴函式值。但是，若仔細觀察 $S(m,n)$ 的樹狀圖，我們會發現有許多的節點是在做相同的事情。越接近樹的底層，重複的節點就越多。因此若有辦法對相同的節點只做一次計算，那麼整體的執行速度不就可以加快了嗎？

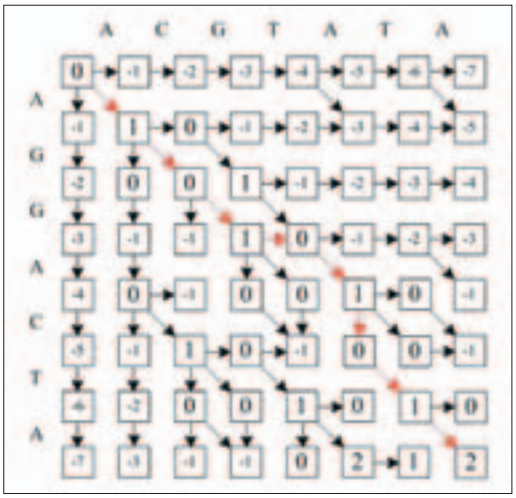
於是我們想到利用由下而上的方式計算 $S(m,n)$：就是對所有相同的節點，只需對第 1 次遇到的節點做計算，然後把計算出來的值儲存起來，待下次碰到同樣的節點時，只要呼叫或參考這



由上而下計算 $S(m,n)$ 的樹狀圖

個節點的儲存值即可，如此便可省去重複計算的時間。

這個主意讓我們可以把上述的樹狀圖簡化成一個（m+1）×（n+1）的二維矩陣圖，其中座標（i, j）的節點等於 S（i,j）。根據遞迴關係，S（i,j）只跟它相鄰的上、左上及左節點有關係，我們必須先知道這3個相鄰節點的分數之後才能夠求出 S（i,j）。其中直線的方向表示 S（i,j）= S（i-1,j）+ δ（a_i ,−），且 a_i 與空白字對齊；斜對線的方向表示 S（i,j）= S（i-1,j-1）+ δ（a_i ,b_j），且 a_i 和 b_j 對齊；橫線的方向表示 S（i,j）= S（i,j-1）+ δ（−,b_j），且空白字和 b_j 對齊。



以比對 AGGACTA 與 ACGTATA 這兩條DNA序列的二維矩陣為例，採用的配分函數是：配對給1分，配錯或開 gap 則給－1分。除此之外，也利用箭頭表示每一個節點的分數是透過哪一個相鄰節點所得到的。從最右下角的節點分數，可以知道 AGGACTA 與 ACGTATA 之間最佳的比對分數是 2分。但是，這兩條 DNA 序列之間的字母該怎麼對齊才能得到這個最佳分數呢？答案是利用在二維矩陣上所留下的箭頭，從最右下角的節點 S（7,7）開始，沿著指進來的箭頭逆推回溯至最左上角的節點 S（0,0）。如此，可以得到一條（甚至數條）由 S（0,0）到 S（7,7）的最短路徑，然後再根據這條路徑上每一箭頭的方向，寫出相對應字母之間對齊的方式，最後再把這些對齊的字母合併起來即可。

A G G - A C T A

A C G T A - T A

AGGACTA與ACGTATA之間最佳的對齊方式

為了求得 S（m,n），我們可以採用以列為導向的方式計算矩陣中每一個節點分數：就是以由左到右的方式先計算出第 1 列節點的分數（開始時令 S（o,o）= 0），然後第 2 列、第 3 列……一直到求出最後一列的 S（m,n）為止。當然也可以採取以欄或斜對角線為導向的方式，計算矩陣中節點的分數。換句話說，整個計算 S（m,n）的過程就好像在表格（即二維矩陣）上填寫數字（即分數）一般。所以，動態規劃的演算法又被稱為填表格的方法。

不難發現，我們總共有（m+1）×（n+1）個節點要計算，而且每一個節點計算最多只花3個加法運算及 2 次的比較運算。因此，整個計算過程所花的運算時間等於 5×（m+1）×（n+1），其中 m 和 n 是影響整個時間複雜度最關鍵的因素，所以整個時間複雜度又被簡寫成 O（m×n），意味著時間複雜度與兩條序列長度乘積成正比。

在生物資訊和計算生物相關的論文中，動態規劃是一種最常見的演算法設計策略，它不僅可以用來解決兩條序列比對的問題，也可以解決許多其他的問題。但是，它並沒有辦法解決所有的問題。正確地說，電腦並不是萬能的！因為數學家早已用理論證明有些問題是電腦沒辦法解決的，例如涂林（Alan Turing）證明停機問題（halting problem）是沒有演算法可以解決的問題。

要如何才能設計出一套有效率的序列比對工具呢？關鍵在於演算法。在生物資訊的論文中，動態規劃是一種最常見的演算法設計策略，它可以用來解決兩條序列比對的問題。

在這種情況下，我們就得嘗試利用其他的演算策略找尋最佳解，例如貪婪策略、分割與擊破策略、消去與搜尋策略和分支與界定策略等等。有興趣的讀者可參考演算法相關書籍。 □

盧錦隆
交通大學生物資訊研究所