The String-to-String Correction Problem

問題定義

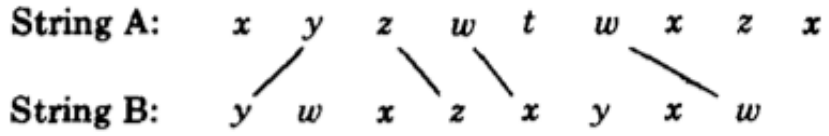
The string-to-string correction problem 探討如何經由一系列的合法操作，將一個字串校正成另外一個字串，然而校正的方法不勝枚舉，故此問題另外要求必須以最少的成本達到目的。本論文中所謂的合法操作包含了以下三種：change（將一個字元更換成另外一個字元）、deletion（刪除一個字元）、insertion（插入一個字元），而每一種操作有各自的成本，整個字串轉換過程的成本定義為所有步驟成本的總和。

以兩個字串 A：fest 和 B：else 為例，並假設三種操作的成本都是一樣的，如此以來，最低的成本意味著以最少的步驟完成校正，校正步驟如下：（1）將 fest 的 f 刪除形成 est。（2）在 est 的 e 跟 s 中間插入 l 形成 elst。（3）將 elst 的 t 更換成 e 形成 else。根據直覺，三個步驟完成校正是此問題的最佳解，然而若要以有系統且有效率的方式解決此問題，則有賴於本論文根據**動態規劃（dynamic programming）**所提出的演算法。

首先定義以下符號：（1）$A\langle i\rangle$：A 字串的第 i 個字元。（2）$A\langle i:j\rangle$：A 字串第 i 個到第 j 個字元所組成的字串。（3）$|A|$：A 字串的長度。（4）**編輯操作（edit operation）** $s_{a\to b}$：將 a 字元更換成 b 字元。當 b 為 Λ（空字元）表示刪除 a；當 a 為 Λ 表示插入 b。（5）**編輯序列（edit sequence）** $S = s_1, s_2, \ldots s_m$：一連串操作所形成的序列。（6）$\gamma(s)$：對應某一個操作所需要的成本。（7）$\gamma(S) = \sum_{i=1}^{m} \gamma(s_i)$：一個編輯序列所需要的總成本。

接著定義兩個字串的**編輯距離（edit distance）**，表示為 δ，也就是將一個字串校正成另外一個字串的最小成本，數學表示如下，給定字串 A、B，則 $\delta(A,B) = \min\{\gamma(S) \mid S$ 是一個可以將 A 校正成 B 的編輯序列$\}$。

Trace 是一種用來表示編輯序列的方法。假設給定字串 A：xyzwtwxzx、B：ywxzxyxw，從 A 到 B 到 trace 圖示如下：



每一條連接 A 到 B 的邊代表一組更換，字串 A 中沒有邊連接的字元會被刪除，而字串 B 中沒有邊連接的字元會被插入。以上圖為例，$A\langle 1\rangle = x$，沒有接上任何邊，代表要將 x 刪除；$B\langle 2\rangle = w$，同樣沒有接上任何邊，代表要將 w 插入；最後，比如 $A\langle 4\rangle = w$，$B\langle 5\rangle = x$ 被一條邊所連接，表示將 w 更換成 x。

Trace 也可以表示成 (T, A, B)，其中 T 是所有邊的集合 $T = \{(i,j) \mid A\langle i\rangle$ 和 $B\langle j\rangle$ 有連接$\}$，以上圖為例，$T = \{(2,1),(3,4),(4,5),(6,8)\}$。可以注意到每個字元至多只能接一條邊，且所有的邊都不允許交叉。對於一個 trace，我們同樣可以定義它的成本如下：

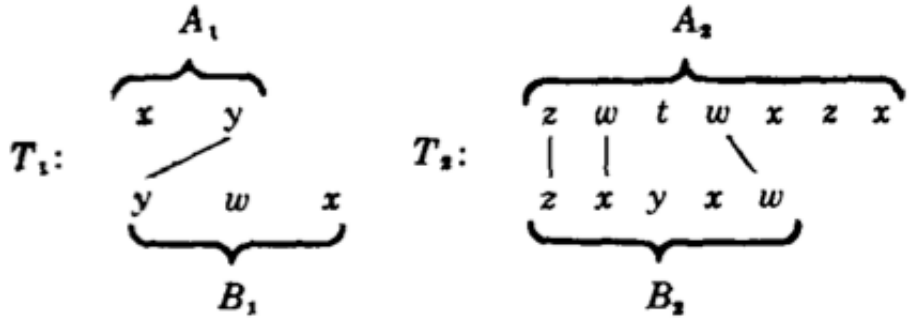
$$\text{cost}(T) = \sum_{(i,j)\in T} \gamma(A\langle i\rangle \to B\langle j\rangle) + \sum_{i\in I} \gamma(A\langle i\rangle \to \Lambda) + \sum_{j\in J} \gamma(\Lambda \to B\langle j\rangle)$$

其中 I 表示 A 中所有沒有被任何邊接上的位置，而 J 表示 B 中所有沒有被任何邊接上的位置，以上圖為例，$I = \{1,5,7,8,9\}, J = \{2,3,6,7\}$。這個公式計算出來的成本，就是對應的編輯序列所需要的成本，而字串 A 跟字串 B 的編輯距離就是所有 trace 中成本最小的值。

解法敘述

在設計動態規劃的演算法有兩個重要的步驟，首先是如何將問題拆解，拆解後，很重要的是確認 principle of optimality 是否成立，以下的解法敘述也會強調這兩個部分。

拆解的過程同樣可以透過 trace 來視覺化，以同樣的例子 A：xyzwtwxzx、B：ywxzxyxw 來做說明。首先，我們將 A 和 B 各自拆解成兩個部分 $A = A_1A_2, B = B_1B_2$，並確保沒有任何邊從 A_1 跨越到 B_2，或從 A_2 跨越到 B_1。又因為所有的邊都不交叉，所以一定可以找到這樣的拆解。下圖是 (T, A, B) 拆解成 (T_1, A_1, B_1) 和 (T_2, A_2, B_2) 的其中一種方法：



這樣的拆解有一個很好的性質：整個 trace 的成本就是兩個 traces 成本的總和，也就是 $\text{cost}(T) = \text{cost}(T_1) + \text{cost}(T_2)$。這個性質告訴我們，如果想要最佳化整個 trace 的成本，可以從最佳化兩個小問題的成本著手。

為了敘述方便，我們另外定義一個函數 $D(i,j) = \delta(A\langle 1:i\rangle, B\langle 1:j\rangle)$, 其中 δ 就是上面定義過的編輯距離，而 $D(i,j)$ 就是將 $A\langle 1:i\rangle$ 校正成 $B\langle 1:j\rangle$ 所需的最小成本。由上面的敘述，我們知道成本可以拆解成兩個小部分考慮，所以可以合理假設：

$$D(i,j) = \min\{D(i-1,j-1) + \gamma(A\langle i\rangle \to B\langle j\rangle), D(i-1,j) + \gamma(A\langle i\rangle \to \Lambda), D(i,j-1) + \gamma(\Lambda \to B\langle j\rangle)\}$$

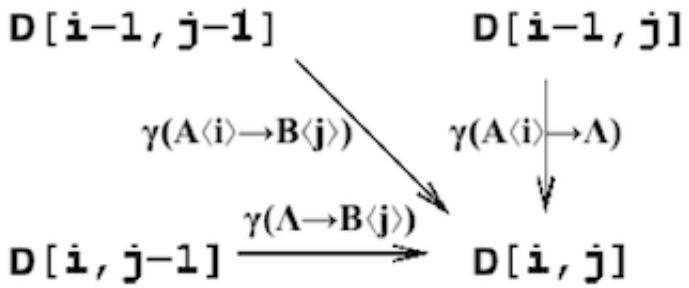
這個公式假設了 principle of optimality 會成立，也就是當我們要求 $D(i,j)$ 的最小成本，拆解後的 $D(i-1,j-1)$、$D(i-1,j)$、或是 $D(i,j-1)$ 也都必須是最小的成本，證明過程大致會是如果 $D(i-1,j-1)$、$D(i-1,j)$、或是 $D(i,j-1)$ 任一不是各自的最佳解，則我們就還可以找到一組解

使得 $D(i,j)$ 更小，故 principle of optimality 必須成立。

動態規劃的成功在於能將大的問題拆解成小的問題，隱含遞迴的概念，並利用空間儲存已經計算過的答案以得到較佳的時間複雜度，而為了不重複計算，必須捨棄 top-down approach 採用 bottom-up approach。首先，針對遞迴的部分討論，任何遞迴還必須給定初始值，後續的計算才能建立在其之上，否則便是無窮迴圈。$D(i,j)$ 的拆解是一個二維的遞迴公式，其初始值可以假設如下：

$$D(0,0)=0\ ;\ D(i,0)=\sum_{r=1}^{i}(A\langle 1:r\rangle\to\Lambda)\ ;\ D(0,j)=\sum_{r=1}^{j}(\Lambda\to B\langle 1:r\rangle)$$

也就是假設第 0 行的初始值就是將 $A\langle 1:i\rangle$ 刪除成空字串的成本，而第 0 列第初始值就是將空字串插入成 $B\langle 1:j\rangle$ 的成本。如此以來，解決此問題之動態規劃就是將如下的二維表格由左上（$i=0\ ;\ j=0$）到右下（$i=|A|\ ;\ j=|B|$）依序填滿：



表格的維度是 $|A|\times|B|$，故時間複雜度也就是走完整個表格的時間 $O(|A||B|)$。

接下來的問題就是如何將編輯序列復原，使用的演算法是對建立完的二維表格由右下至左上執行 backtracking，每走一步必須決定一次方向，步驟如下：

1. 從右下角出發，將 i 和 j 分別初始化為 $|A|$ 和 $|B|$
2. 當還沒走到最左上角 $i\neq 0$ 且 $j\neq 0$ 時：
   2.1 如果 $D[i,j]=D[i-1,j]+\gamma(A\langle i\rangle\to\Lambda)$，則往上走一格 $i:=i-1$
   2.2 否則如果 $D[i,j]=D[i,j-1]+\gamma(\Lambda\to B\langle j\rangle)$，則往左走一格 $j:=j-1$
   2.3 否則 print$((i,j))$，並往右上角走一格 $i:=i-1\ ;\ j:=j-1$

往上一格，表示此步驟的 trace 是 $A\langle i\rangle\to\Lambda$，也就是將 $A\langle i\rangle$ 刪除。往左一格，表示此步驟的 trace 是 $\Lambda\to B\langle j\rangle$，也就是將 $B\langle j\rangle$ 插入。最後，往右上走一格，表示在 trace 表示法中，$A\langle i\rangle$ 和 $B\langle j\rangle$ 有一條邊連接，也就是將 $A\langle i\rangle$ 更換成 $B\langle j\rangle$。根據數學歸納法，每一個逆推步驟都是正確的，最後所得到的 trace 也會是正確的編輯序列。

填表時，每一格必定由左上、左、上方的其中一格過來，故 backtracking 也必然可以沿著此路徑走回最左上角 $i=0$ 且 $j=0$，迴圈必然終止。每次迴圈必然往左上、左、上方擇一而走，故到最左上角，最多只會走 $|A|+|B|$ 步，也就是最多只會執行 $|A|+|B|$ 次迴圈，故時間複雜度為 $O(|A|+|B|)$。

讀後心得

Longest common subsequence (LCS) 可以視為 string-to-string correction problem 的一個特例。假設 insertion 和 deletion 的成本都為 1，mismatch 的成本為 2，根據這樣的設計所找出來的 trace，會使得每一條邊都連接相同的字元，而且長度是最長的，而最小的成本為 $|A| + |B| - 2 \times |\text{LCS}|$。課堂中的演算法是利用動態規劃直接最大化（maximize）LCS 的長度，與本論文所提到的演算法：最小化（minimize）成本，雖殊途而同歸，且時間複雜度相當：建表需要 $O(|A| \times |B|)$，逆推需要 $O(|A| + |B|)$。

在閱讀本論文時，雖然未看到作者曾經提到 dynamic programming 或是 principle of optimality 等片語，但是在演算法正確性的推導過程中，一而再、再而三地使用到其概念，心中有疑惑，動態規劃的模型在當時是否已經發展成熟？求證後發現，動態規劃最早在 1940 年代已經被 Richard Bellman 提出，到了 1974 年，也就是本論文發表的時候，相關概念理應發展成熟，或許動態規劃的概念對作者來說已經是一種直覺，甚至不需在文章多作說明。

最後，想舉例 string-to-string correction problem 在現代基因工程的應用，這樣的應用在本論文發表當時仍是無法想像的。首先是基因定序，必須將基因組（genome）先分段（fragmentation），才能分別對每一個段落作定序，然而要將分段後的結果重組回原先完整的序列，就有賴於以動態規劃為基礎的演算法來計算編輯距離，將不同片段前後剪剪貼貼找出最能吻合的方式拼湊起來（註：還需搭配圖論相關演算法才能完成基因定序）。另外，LCS 也常利用於物種分類，目標是求出同一段基因或是蛋白質序列的相似程度來推論其演化關係。在這類問題，change、insertion 和 deletion 的成本會因應每個核苷酸（neucleotides）或胺基酸（aminoacids）有不同的更換成本，因而產生所謂的取代矩陣（substitution matrix），又長序列的 insertion 和 deletion 在生物基因體相當常見，所以成本往往會設計地較低，以反映真實演化的過程。最後，相當有幸可以讀完這篇原文，看到這個問題最初如何被構思的過程與歷史。