錯誤更正碼簡介

趙啓超演講

李啓鈴記錄

今天很高興來跟大家介紹 「錯誤更正碼」, Error–Correcting Codes, 通常縮寫成 ECC。在日常生活中, 我們經常可以遇到 ECC 的應用, 譬如在買電腦的時候, 老闆說他的 RAM 是 ECC RAM, 也就是說他的 RAM 有錯誤更正碼在裡面。這錯誤更正碼到底是做什麼用? 通常是應用在通訊的過程中, 實際上通訊分很多種, 平常想像到的是在空間中做通訊, 譬如從地球的某一點傳資料到地球的另外一點。另外還有一種通訊的情況, 就是在時間上做通訊, 像平常看到電腦裡的磁帶 (magnetic tape)、磁碟 (magnetic disk); 又譬如雷射唱片 (compact disk, CD) 也是, 它在製造的時候把音樂寫進去, 然後再把它放到雷射唱盤 (CD player) 讀出來, 這也是一個在時間上做通訊的例子。通常我們可以把通訊過程變成下面這個模型 (model) (如圖 1)。

圖 1

就是說有一些資料 (data) 經過一個東西叫做通道 (channel), 因爲通道會有許多干擾或雜訊, 比方說你把音樂寫在 CD 片上, 不小心刮了一塊, 讀出來的時候那地方就錯了, 所以, 經過通道之後, 就會變成有雜訊的資料 (noisy data), 而基本上錯誤更正碼就是要對付雜訊或干擾, 把錯誤改正回來。

一. 錯誤更正碼的由來

爲什麼會想到利用錯誤更正碼? 1948 年以前, 大家在做通訊系統的時候, 都只想到要如何使訊號接收得比較好, 比如把天線加大, 或者把訊號傳遞的功率加大, 但這些都是在物理上做改進。直到 1948 年 Claude Shannon 寫了一篇論文叫做 A mathematical theory of communications, 他說事實上我們不必這樣做, 很多情形只要在資料上做手脚, 只要傳資料的速率小於每個通道特有的量, 術語叫做通道容量 (channel capacity), 就一定有辦法讓資料錯誤的機率很小, 要多小都可以, 而要達到這樣的碼 (code) 一

定會存在, 但是他並沒有說用什麼樣的碼, 只證明了一定會有。不過, 因爲他提出了這件事, 後來造成了整個領域的發展, 這個領域叫做訊息理論 (information theory), 其中有很重要的一部分就是 ECC 的理論。從1948年到現在40幾年來, 這整個領域的發展就是因爲 Shannon 那時候的貢獻。

基本上, Shannon 說什麼呢? 原來我們是從資料到通道這樣直接過來, 現在我們讓資料在傳出前先經過一個編碼器 (encoder)(如圖2)。

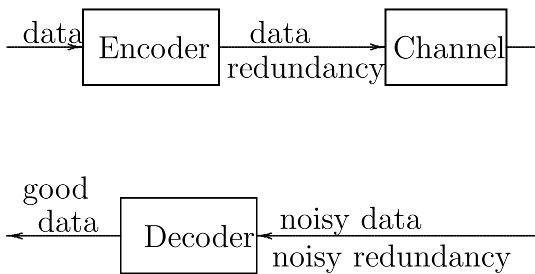

圖 2

經過編碼器之後, 傳出來的東西除了原始的資料之外, 又加了一些多餘的資料 (redundant bits), 使得裡面有一個數學的結構存在, 當經過通道以後不論是原始的資料或多餘的資料都可能有錯, 可是因爲它裡面有數學結構, 就可以經由解碼器 (decoder), 依原有的數學結構把錯誤的東西改回來, 這就是錯誤更正碼的基本構想。

二.Single–Error Correcting(SEC) Codes

我們現在先來看一個簡單的例子, 有三個集合 (sets) A、B、C 兩兩相交 (如圖3.1)。

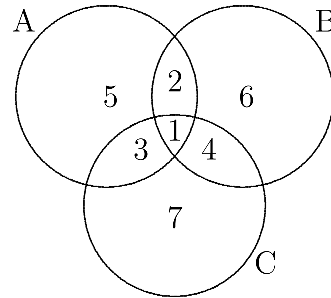

圖 3.1

我們把每一小塊編號爲$1, 2, \cdots, 7$, 然後假設原來傳 1101, 依序把它寫在編號$1, 2, 3, 4$的空格上, 然後多加三個 bits 上去, 讓每個圓圈裡面1的個數是偶數, 就如圖3.2。

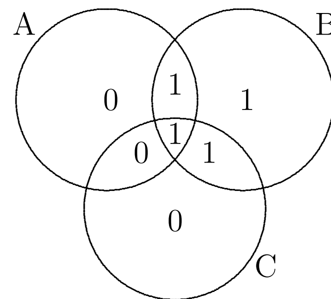

圖 3.2

所以加進去的三個 bits 是010, 最後要傳的是1101010, 這就是剛才講的, 要多加一些東西讓它們有數學結構, 這樣就可以改正錯誤。現在看下面的例子, 原來傳的是1101010, 假設第6個 bit 錯了, 1變成0, 如圖3.3a。

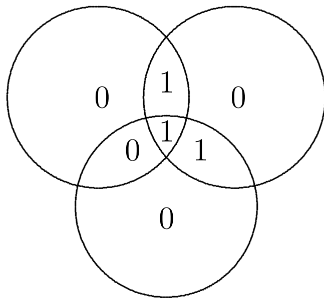

圖 3.3a

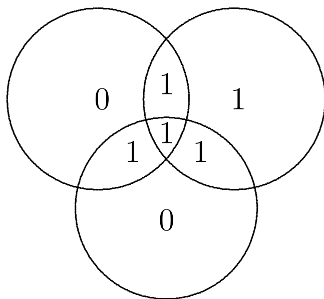

圖 3.3b

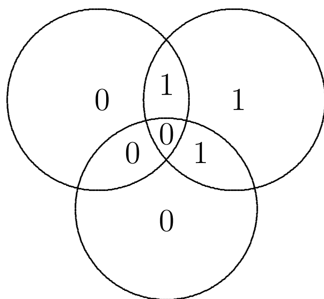

圖 3.3c

解碼器這邊知道它原來的數學結構, 就是每個圓裡面1的個數是偶數, 所以 A 有兩個1, 沒有問題, C 有兩個1, 也沒有問題, 可是 B 有3個1, 有問題, 因此我們知道是6這一塊錯了, 就把這個0改回來變成1, 以符合原來的結構。再舉一個例子, 假如是錯第3個 bit, 其他都對, 如圖3.3b, 那麼 A 有3個1, 有問題, B有4個1, 沒有問題, C 有3個1, 也有問題, 所以我們就找$A \cap C \cap B^C$這一塊, 也就是3這一塊錯了, 就把它改回來。同樣道理, 如果是錯在第一個 bit, 如圖3.3c, 三個圓都有問題, 所以你知道是錯在$A \cap B \cap C$這一塊, 然後把它改回來就變成1。這裡雖然只講了三個情形, 但其他的情形, 都只是對稱而已, 所以這個碼可以改一個錯, 我們就稱做 single–error correcting (SEC)。這個碼有一個術語叫做 (7, 4) Hamming code, 爲什麼叫 (7, 4)? 就是全部長度是7, 而只有4個 bits 是原來要傳的資料, 所以叫 (7, 4)。那爲什麼叫 Hamming code? 因爲這個碼是 Hamming 在1950年所發明的。

那麼, 假如有兩個錯會怎麼樣? 比方說, 如果是6、7兩個 bits 錯了, 如圖3.4。

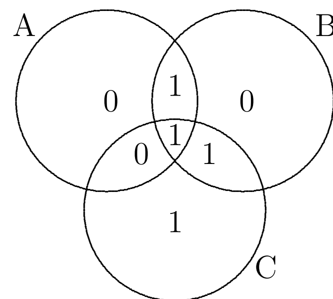

圖 3.1

那麼 A 不會有問題, B 和 C 都有問題, 照剛才的方法就要改4這個 bit, 結果除了原來的兩個錯, 反而又多出了一個錯, 事實上, 我們可以證明任何兩個 bits 錯, 如果照原來的方法去改, 都會再多出一個錯, 爲什麼? 因爲這個碼不能改兩個錯, 最多只能改一個錯, 這跟我們加了多少個多餘的 bits 有關, 在這個例

子我們多加了3個 bits, 如果要改兩個或兩個以上的錯, 就要多加新的 bits。這就是一般所謂天下沒有白吃午餐的道理, 要想得到很大的收穫, 就要付出相對的代價。

1. (7, 4) Hamming Code 的數學結構

剛才我們看到了 (7, 4)Hamming code 的例子, 那爲什麼它能改一個錯呢? 首先我們定義一個二進位的運算 (binary arithmetics), 包含加法和乘法, 列在表一。

	0	1
0	0	1
1	1	0

加法

	0	1
0	0	0
1	0	1

乘法

表 1

這基本上就是所謂 modulo 2的運算, 先加起來或乘起來後再除以2取餘數。定義這樣的運算之後, 剛才的圓圈所要滿足的式子就可以寫下來, 如下式:

$$\begin{cases} x_1 + x_2 + x_3 \quad + x_5 = 0 \\ x_1 + x_2 \quad + x_4 \quad + x_6 = 0 \\ x_1 \quad + x_3 + x_4 \quad + x_7 = 0 \end{cases}$$

$$x_i \in \{0, 1\}$$

所以剛才的限制, 基本上變成數學上的一組線性方程式 (linear equations), 那所有的 codewords 是什麼? 我們如果把這組線性方程式的係數寫成一個矩陣

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix},$$

令C代表所有 codewords 所構成的集合, 則$\underline{x} \in C$ 且$\underline{x}$是row vector, 就必須滿足$H\underline{x}^T = \underline{0}$, 因此整個$C$就是矩陣$H$的 null space, 它的 dimension 就等於7減掉這個矩陣的 rank 3, 也就是4, 所以C集合裡面就有$2^4 = 16$個 codewords。從另外一個觀點來看, 我們隨便給4個 data bits, 然後將其編碼成7個 bits, 因這4個 bits 可以隨便選, 故有2^4等於16種可能, 所以全部共有16個 codewords.

1.1 Syndrome

我們現在知道所有 codewords 構成的集合是H的 null space, 那要怎麼去解碼呢? 當一個 codeword $\underline{x}$傳過通道之後會錯, 我們就假設成加上一個錯誤向量$\underline{e}$, 如圖4。

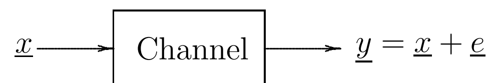

圖 4

比方說第一個 bit 錯了, 就表示$\underline{e}$的第一個bit是1, 因爲如果原來是0, 加1會等於1, 原來是1, 加1會等於0, 所以加上這個$\underline{e}$ 就可以表示錯誤的部分。接著我們介紹一個東西叫做syndrome, syndrome 就是指徵狀的意

思，傳出來的東西要知道錯在那裡，就要算一下這個東西，看它到底有什麼徵狀，然後才能做判斷。這個 syndrome 怎麼算？ 就是把H乘上$\underline{y}^T$，現在$\underline{y} = \underline{x} + \underline{e}$，利用分配律展開，得到$H\underline{x}^T + H\underline{e}^T$，因爲$\underline{x}$是codeword，所以$H\underline{x}^T = \underline{0}$，最後剩下$H\underline{e}^T$，所以 syndrome 只跟錯誤向量有關，跟傳那個 codeword 無關，故可以從 syndrome 把$\underline{e}$猜出來，可是這可以有很多解，比方說$\underline{e}$是一個解，隨便再加上一個codeword 也是一個解，所以會有$2^4 = 16$個解，這麼多解我們取那一個呢？ 通常是取裡面1的個數最少的那一個，也就是錯的個數最少，因爲在一般的情況下會錯的機率比較小，不會錯的機率比較大，所以就統計上來說，找一個錯的個數最少的向量，表示它可能的機率最大，因此就可從得到的$\underline{y}$把原來的$\underline{x}$算出來。舉個例子來說，$\underline{y} = (1101000) = (1101010) + (0000010)$，則 syndrome 是

$$\begin{aligned}\underline{s}^T &= H\underline{y}^T \\ &= \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix},\end{aligned}$$

那它原來的$\underline{e}$是什麼？ 也就是假設$\underline{e}$裡面的某一個bit 是1，與H乘出來的結果是$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$，在這個例子裡$\underline{e}$ 就是(0000010)，但這只是一個解，還有其他15個解，但是這個解是機率最大的，也就是1的個數是最少的，所以解碼回來後，可得出(1101010) 就是我們原來傳的 codeword。

爲什麼這個碼可以改一個錯？因爲如果沒有錯，我們去算 syndrome 得到$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$，如果第一個 bit 錯了，算出來是$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$，第二個 bit 錯了是$\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$，$\cdots$，第七個 bit 錯了就是$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$，這些 syndromes 全部都不一樣，所以每一個 syndrome 就對應到一種錯的情況，我們只要知道它的徵狀是什麼，就可以知道錯在那裡，然後把它改正過來，這就是 (7, 4) Hamming code 可以改一個錯的原理。

1.2. Hamming Distance & Hamming Weight

剛才我們是從 syndrome 的觀點來看，現在從另外一個觀點 Hamming distance 跟 Hamming weight 來討論。

什麼是 Hamming distance？ 例如說兩個向量$\underline{x} = (0001011)$ 和$\underline{x}' = (1101010)$，Hamming distance 就是它們不一樣的地方有幾個，這個例子$d_H(\underline{x}, \underline{x}') =$

3, 因爲它們之間有3個 bits 不同。而 Hamming weight, 是指一個向量不是$\underline{0}$的的地方有幾個, 所以$w_H(\underline{x}) = 3, w_H(\underline{x}') = 4$。由這個定義可以推到$\underline{x}$ 和$\underline{x}'$的Hamming distance 事實上只是$\underline{x} + \underline{x}'$的Hamming weight, 因爲$0+0=0, 1+1=0$, 只有兩個不同時加起來才會等於1, 所以加起來之後不爲0 的 bits 的個數, 就等於它們之間不同的 bits 的個數, 也就是 Hamming distance。有了這些定義以後, 我們可以定義一個碼的 minimum distance, 就是任何在這個碼中的兩個不同的 codewords 的 Hamming distance, 它們之中最小的值就是這個碼的 minimum distance:

$$d_{\min}(C) = \min_{\substack{\underline{x},\underline{x}' \in C \\ \underline{x} \neq \underline{x}'}} d_H(\underline{x}, \underline{x}').$$

我們還可以定義 minimum weight, 就是所有不爲0的 codewords, 它的 Hamming weight 的最小值就是這個碼的 minimum weight:

$$w_{\min}(C) = \min_{\substack{\underline{x} \in C \\ \underline{x} \neq \underline{0}}} w_H(\underline{x}).$$

一個碼的 minimum distance 和 minimum weight 乍看並不一樣, 但實際上因爲這些碼都是線性 (linear) 的, 這是因爲 null space 是一個 linear subspace, 所以, 任何兩個 cordwords 加起來還是一個 codeword, 而兩個 codewords 的 Hamming distance 會等於相加後的 Hamming weight, 所以對線性碼來說, $d_{\min}(C) = w_{\min}(C)$。從這個觀點來看, 剛剛這個碼是

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

的 null space

那它的 minimum distance 是多少? 因 minimum distance 就是 minimum weight, 所以我們從一個 codeword 除了全部是0之外, 它的1的個數最少有幾個來考慮。有沒有可能是1? 不可能, 因爲如果有一個 codeword 的 weight 是1, 那H乘上去之後會得到某一個 cloumn, 絕不會是$\underline{0}$, 所以這個codeword 就不會在 null space 裡面, 因此不可能是1。2也是不可能的, 因爲一個 weight 是2的 codeword 與H相乘後, 會變成H中兩個不同的 column 相加, 但是H中的每一個 column 都不一樣, 所以相加之後不可能是$\underline{0}$, 所以這樣的codeword 也不在 null space 裡面。那可不可能是3? 答案是對的, weight是3的 codeword 與H相乘變成3個 columns 相加, 考慮譬如第一個 column 加第二個 column 等於$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$是第七個 column, 所以有一個 codeword 是 (1100001), 它的 weight 是3, 而且很簡單地可以算出這樣的 codewords 總共會有7個, 因此這個碼的$d_{\min} = w_{\min} = 3$。

另一方面, 假設一個碼要改t個錯, 它會與什麼有關? 它跟 minimum distance 有關, 就是說如果要改t個錯, 它的 minimum distance 至少要$2t+1$, 爲什麼? 我們從圖5來看。

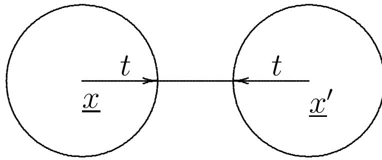

圖 5

有兩個球, 在球中心是一個 codeword, 這球包含所有跟這個 codeword 的 Hamming distance 小於等於t的二位元向量 (binary vectors), 因爲要改t個錯, 這兩個球不能碰在一起, 假如碰在一起, 那會有一點與$\underline{x}$的距離小於等於t, 距離$\underline{x}'$ 也小於等於t, 這時候就不知道要把它改成$\underline{x}$, 或者是$\underline{x}'$, 因此就不能改t個錯, 所以這兩球的距離至少是1, 交集是空集合, $\underline{x}$和$\underline{x}'$的距離至少要是$t+1+t=2t+1$, 任何兩個 codewords 的距離至少都要$2t+1$, 所以 minimum distance 也一定至少要$2t+1$。反過來說, 如果 minimum distance 大於等於$2t+1$, 當發生錯誤的個數小於等於t時, 對任一個接收到的向量, 我們可以找到跟它距離最小的唯一一個 codeword 把它改回來, 因此這是若且唯若的條件。所以要能改t個錯, 隨便取兩個不同的 codewords, 它們不一樣的地方至少要有$2t+1$個。所以剛才那個碼的 minimum distance 是3, 從這個觀點來看, 它可以改一個錯, 而且只有一個。

2. 一般情形的 SEC Hamming Codes

我們看前面 (7, 4) Hamming code 的例子, 它可以改一個錯是因爲它的 minimum distance 等於 3, 如果從H的 columns 的角度來看, 是因爲它每個 columns 都不一樣, 假設有兩個 columns 一樣的話, 就一定會有一個 weight 是2 的 codeword, 經與H相乘, 讓這兩個 columns 加起來, 就變成$\underline{0}$了, 所以每個column 都要不一樣, 可是我們當然希望有愈多 columns 愈好, 爲什麼? 因爲這三個 rows 表示加了三個 bits, 有愈多的 columns 表示同樣加三個 bits 而 data 可以傳得愈多, 因此, 我們希望 rows 愈少愈好, columns 愈多愈好, 在這兩個前提下, 我們看先前這個例子, 它有3個 rows, 所以 columns 最多可以有7個, 爲什麼不能有8個? 因爲假設有8個, 而且每一個 column 都不一樣, 那有一個一定會是$\begin{pmatrix}0\\0\\0\end{pmatrix}$, 但不能有$\begin{pmatrix}0\\0\\0\end{pmatrix}$, 因爲假設有, 又剛好錯在那個地方, 則得出來的 syndrome 是$\begin{pmatrix}0\\0\\0\end{pmatrix}$, 就跟沒有錯一樣了, 這個 bit 的錯就改不掉, 所以說, 假如有3個 rows 的話, 最多可以有$2^3-1=7$個不一樣的 columns。在一般的情況又是怎樣? 假如 rows 有m個, 最多可以有2^m-1個 columns, 每個都不一樣, 如下式:

$$H=\underbrace{\begin{pmatrix}1 & 0 & & 1\\ 0 & 1 & & 1\\ \vdots & 0 & \cdots\cdots & 1\\ \vdots & \vdots & & \vdots\\ 0 & 0 & & 1\end{pmatrix}}_{2^m-1}\left.\vphantom{\begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix}}\right\} m$$

所以這個碼全部有2^m-1個 bits, 然後這個H的 rows 都是線性獨立 (linearly inde-

pendent), 因這個碼是H的 null space, 它的 dimension 是2^m-1減掉它的 rank m, 故這是一個$(2^m-1, 2^m-m-1)$ SEC Hmmaing code。

三.Double-Error Correcting (DEC) Codes

前面講的碼只能改一個錯好像不大好, 因為有時候可能會有兩個錯, 雖然通常錯兩個的機率比錯一個的機率小很多, 但是通道情況很差的時候, 還是會發生, 如果用剛才的碼, 像前面的例子, 就會多製造錯誤出來, 反而比沒有用碼更糟, 那怎麼辦? 所以我們就要用可以改兩個錯的碼。

1. 由 SEC 到 DEC

要怎麼改兩個錯? 我們先來看看原來改一個錯的時候是怎麼做的, 現在考慮$m=4$的情形,

$$H=\begin{pmatrix} 0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1 \\ 0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1 \\ 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1 \end{pmatrix},$$

這是一個4×15的矩陣, 每一個 column 都不一樣, 是按二進位的順序來排列, 這個碼是 (15, 11) SEC Hamming code, 有4個 bits 是多加的, 11個 bits 是眞正的 data, 就是說加了4個 redundant bits 可以改一個錯, 很明顯我們會想到是不是再多加4個 bits 就可以改兩個錯, 也就是再多加4個 rows, 如果取得好的話就可以改兩個錯, 這是一種直覺上的想法, 那麼, 現在我們來試試看到底可不可以, 我們用新的符號來簡化H

$$H_{SEC}=(\underline{\alpha_1}\underline{\alpha_2}\cdots\underline{\alpha_{15}}),\ \text{where}\ \underline{\alpha_1}=\begin{pmatrix}0\\0\\0\\1\end{pmatrix},\underline{\alpha_2}=\begin{pmatrix}0\\0\\1\\0\end{pmatrix},\cdots,\text{etc.}$$

然後再把向量的標號拿掉, 只要不會混淆就沒關係, 如果再加4個 rows, 底下也同樣會有4×1的 column vectors, 用β_i來表示, 所以

$$H_{DEC}=\begin{pmatrix}\alpha_1 & \alpha_2 & \alpha_3 & & \alpha_{15} \\ & & & \cdots & \\ \beta_1 & \beta_2 & \beta_3 & & \beta_{15}\end{pmatrix}$$

我們希望這些β_i取好一點, 以便可以改兩個錯。我們現在來看 syndrome, 當沒有錯的時候, 它是$\begin{pmatrix}0\\0\\\vdots\\0\end{pmatrix}$, 有一個錯的時候, 得到結果是某一個 column, 兩個錯的時候, 得到的是兩個不同 columns 的和, 現在要改兩個錯, 所以我們希望這些 syndromes 全部都不一樣, 就是每個 column 及任何兩個 columns 相加的和都不一樣。假設我們這麼想, 把β_i寫成α_i的函數, 也就是$\beta_i=f(\alpha_i)$, 再看看能不能達到這個目的, 我們可以選許多種函數來試, 看前面的條件能不能成立, 譬如f用線性的或者是平方, 但都不行, 結果是選了一個三次方的就可以, 但是你現在會問, 這α_i明明是向量, 要怎麼取三次方? 所以我們必須知道要怎麼樣來乘這些向量。

2. Finite Fields

要怎麼樣乘向量？比如說剛才的 $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ 和 $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$ 要怎麼乘？乘出來還要是一個向量，事實上這是可以辦到的, 因爲這全部的4×1的 columns 一共有16個不同的 binary vectors, 這16個向量我們可以把他們變成一個代數上叫做 field 的東西, 如果是 field 的話, 我們就可以做加減乘除, 事實上除了乘法之外, 我們還要有乘法反元素, 所以我們需要把他們變成一個 field。假如你忘了 field 是什麼, 這裡稍微複習一下, field的定義基本上很簡單, 就是可以做加減乘除的運算, 也就是說有一個集合有兩個運算, 加法和乘法, 一方面要求對加法是一個 Abelian group, 就是說要有封閉性、結合性、加法單位元素, 還有加法反元素, 而且是可交換的; 那麼對於乘法, 除了0之外, 也構成一個 Abelian group; 除了這兩個條件之外, 乘法對加法的分配律也要成立, 這樣就構成一個 field。剛才我們要的是16個元素的 field, 如果把16個元素叫做$0, 1, 2, \cdots, 9, A, B, C, D, E, F$, 然後看表2, 這裡兩個表分別表示加法和乘法, 我們可以驗證一下, 它對加法是一個 Abelian group, 乘法除了0之外也是一個 Abelian group, 分配律也成立, 所以確實是一個 field, 可是你會問這東西是怎麼得到的?

Hexadecimal field

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
2	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
3	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
4	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
5	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
7	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
B	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
F	0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	A

表 2

假如p是一個質數, $Z_p = \{0, 1, 2, \cdots, p-1\}$, 它對 arithmetics modulo p是一個 field。通常會不是 field 的原因出在那裡? 通常問題就是乘法反元素, 但p是質數

的話, 那麼$1, \cdots, p-1$一定有乘法反元素, 要怎麼算? 這可以用 Euclid′s algorithm, 就是所謂的輾轉相除法把它算出來。現在看另外一個問題, 如果是一個 finite field, 就是說一個元素個數是有限的 field, 它的元素個數是不是任意的? 譬如說有沒有一個元素個數是10 的 field? 事實上, 一個 finite field 的元素個數一定要是質數的次方才行, 10就不是, 所以不會存在一個元素個數爲10的 field, 再看$16=2^4$ 所以有可能, 又比方$4=2^2$也有可能, 可是你要做的對才行, 譬如Z_4, 它的元素個數是4, 但是它不是一個 field, 因爲有些情況沒有乘法反元素, 比方說2怎麼乘都是偶數, modulo 4之後仍然是偶數, 永遠都不會等於1, 因此不會有乘法反元素, 所以對有4個元素的集合你有辦法把它變成 field, 但做法不對就不行, 如果是16的話, 就不能用Z_{16}, 那不是 field, 很多情況會沒有乘法反元素, 那麼要怎麼做呢? 這裡先定義一個符號, 因爲 finite field 的元素個數一定是質數的次方, 所以一個有p^m個元素的 finite field, 我們通常稱爲$GF(p^m)$, 這是爲了紀念法國數學家 Galois, 有很多 finite field 上的理論是他發明的, 不過很不幸他在20幾歲就去世了。現在我們要來建構一個有p^m個元素的 finite field, 首先把每一個元素看成一個多項式, 它的係數是在Z_p裡面, 係數從0到$p-1$, 然後它的次數小於m, 最多到$m-1$, 因此就有m個位置, 每個位置可以是0到$p-1$, p個選擇, 總共就有p^m個元素, 而運算時就照平常多項式的加法、乘法一樣, 係數則照Z_p的運算, 只不過要再 modulo 一個$a(x)$, $a(x)$是一個次數是m 的 irreducible polynomial, irreducible polynomial 是說最多只能提出一個常數, 不能把它分解成低次項的乘積, 這樣運算後再 modulo $a(x)$就可以建構一個 field 出來, 這種概念和Z_p中的做法很類似, 這個 irreducible polynomial 就相當是質數的地位。現在舉個例子來看, 比方說剛才表2所列的那個16進位的一個 field, 取$a(x)=x^4+x+1$, 這個在 binary 的 polynomial 不能再分解了, 若假設是x^4+1, 這可以分解成$(x^2+1)^2$, 因爲把它展開就變成$x^4+x^2+x^2+1$, 但$x^2+x^2=0$, 所以就變成x^4+1, 這跟我們平常在實數裡的運算不大一樣, 在二進位的世界裡面會比較奇怪一點。但x^4+x+1就不能分解, 是一個 irreducible polynomial。我們把剛剛表2中的元素$0,1,2,\cdots,F$用 binary 表示, 0看成0000, 1看成0001, 2看成0010, 3看成0011, $\cdots, F$ 看成1111, 把它對應到 polynomials 就變成$0\to 0, 1\to 1, 2\to x, 3\to x+1, \cdots, F\to x^3+x^2+x+1$, 舉個例子, A是1010, C是1100, $A+C=0110$, 也就是6, 我們看剛才的表2中的$A+C=6$就是這樣來的, 再看$A\cdot C=?$ A是x^3+x, C是x^3+x^2, $A\cdot C=x^6+x^5+x^4+x^3$, 但要把它變成次數是3的多項式, 所以要 modulo x^4+x+1, 最後答案就是1, 再看剛才的表2, $A\cdot C=1$沒有錯, 因此基本上所有元素個數是p^m的 finite fields 都可以這樣建構出來, 而所有元素個數是p^m的 fields 跟用這種方法建構出來的 field 都是 isomorphic, 只是名字不同, 但結構上是完全相同, 所以如果

isomorphic 的都不計, finite field 是唯一由它的元素個數所決定, 也就是$GF(p^m)$是唯一的。在實際應用上, 有不少人在設計特別的電路來做 finite field 裡面的運算, 加法的電路很簡單, 乘法就比較麻煩一點, 因爲要節省計算的時間, 這種做乘法的電路叫做 finite field multiplier, 有不少方法可以減少電路及計算時間的複雜度, 基本原理是把每一個元素表示成一組基底 (basis) 的線性組合, 取不同的基底運算的複雜度就不一樣, 取一個好一點的基底, 就可以算得比較快。

3. BCH Codes

我們回到原來的主題, 矩陣H_{DEC}上半部的 columns 是從 $\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$ 開始, 一直到 $\begin{pmatrix}1\\1\\1\\1\end{pmatrix}$, 我們用剛才的表示方法是$1, 2, 3, \cdots, F$, 下半部的 columns 就是它的三次方, 用剛剛講過的這個過程來算, 比方說$C^3 = (C \times C) \times C = F \times C = 8$, 同樣的道理, 其他項也可用一樣的方法計算得到。因此

$$H_{DEC}=\begin{pmatrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & A & B & C & D & E & F\\ 1 & 8 & F & C & A & 1 & 1 & A & F & F & C & 8 & A & 8 & C\end{pmatrix},$$

這是一個 (15, 7) 的碼, 因爲一共有15個 columns, 有8個 rows, 這8個可以證明都是線性獨立的, 所以這個碼的 dimension$= 15 - 8 = 7$, 原來有7個 bits 是 data, 另外8個 bits 是加進來的, 這個碼我們叫做 (15, 7) BCH code, BCH是三個人姓名 的第一個字母, 這個碼原來叫做 BC code, 是由 Bose 和 Ray–Chaudhuri 兩個人在1960年所發明, 後來又發現在1959年法國的一個期刊上有另外一個人叫 Hocquenghem, 也發明同樣的碼, 於是就改稱做 BCH code。

這個碼的 minimum distance 可以證明出來是5, 因此可以改兩個錯, 我們看怎麼改? 跟前面一樣, 我們接收到一個向量, 把它乘上矩陣H之後, codeword的部分就變成$\underline{0}$, 只剩下H與$\underline{e}$相乘的結果, 這部分原來是一個8×1的向量, 我把它上面4×1 的向量和下面4×1的向量寫成$GF(16)$中的兩個數s_1和s_3, 因此$\underline{s}^T = H\underline{e}^T = \begin{pmatrix}s_1\\s_3\end{pmatrix}$, 現在要從 syndrome 來改錯, 假如知道錯兩個錯, 令它是x是y兩個 bits, 則$\underline{s}^T = \begin{pmatrix}x\\x^3\end{pmatrix} + \begin{pmatrix}y\\y^3\end{pmatrix} = \begin{pmatrix}x+y\\x^3+y^3\end{pmatrix} = \begin{pmatrix}s_1\\s_3\end{pmatrix}$, 因此, 得到兩個關係式$x + y = s_1$和$x^3 + y^3 = s_3$, 再把它簡化, 先取$s_1^3 = (x + y)^3 = x^3 + x^2y + xy^2 + y^3$, 再加上$s_3$, $s_1^3 + s_3 = x^2y + xy^2 = xy(x + y)$, 然後除以$s_1$, 得到$\frac{{s_1}^3+s_3}{s_1} = xy$, s_1和s_3是我們已經知道的, 所以可把$x + y$ 和xy算出來, 令$\sigma_1 = x + y, \sigma_2 = xy$, 知道$x + y$和$xy$就可以把$x$、$y$ 算出來, 因爲這個和一元二次方程式的解很像, 這裡正跟負是一樣的, 所以就是去解這個方程式:

$$\theta^2 + \sigma_1\theta + \sigma_2 = 0,$$

解出來的兩根就是x和y。剛才假設錯兩個, 如果只錯一個會怎麼樣? 錯一個的話, 就只有

一個x, 沒有y, 所以$s_3 = s_1^3, s_1^3 + s_3 = 0$, 故$\sigma_2 = 0$, 因此方程式有一個根是0, 這表示只有一個錯, 這樣子這個碼也可以改一個錯, 所以無論是沒有錯, 錯一個或錯兩個的情形都可以解出來。

舉個例子來說, 假設 syndrome $\underline{s}^T = \begin{pmatrix} 8 \\ 3 \end{pmatrix} = \begin{pmatrix} s_1 \\ s_3 \end{pmatrix}$, 計算$\sigma_1 = s_1 = 8$, $\sigma_2 = \frac{{s_1}^3+s_3}{s_1} = \frac{8^3+3}{8} = \frac{A+3}{8} = \frac{9}{8} = 9 \cdot 8^{-1} = 9 \cdot F = E$, 因此要去解$\theta^2 + 8\theta + E = 0$這個一元二次方程式, 這裡所有的運算都是照表2的加法和乘法來做, 那要怎麼來解? 或許有人代公式$\frac{-b\pm\sqrt{b^2-4ac}}{2a}$, 可是這裡不行, $2a$就等於0了, 所以基本上這沒有公式可代, 那我們就用"Try and Error"的方法, 爲什麼我們可以這麼做而平常在解實數或複數根的時候就不可以呢? 因爲這個 field 是有限的, 只有16個數而已, 因爲這個方程式有兩個根, 最多試15次就可以了, 若對$GF(2^8)$裡的數, 則最多試255次就知道了, 也許你會說255好像很多, 事實上在電路上做這個運算是很快的, 255次算是很少的。我們現在就來試, 比方說$\theta = 1$代進去, 變成$1 + 8 + E = 7$, 不對, 再試2, $4 + 3 + E = 9$, 也不對, 再試3, $5 + B + E = 0$, 對了, 所以3是一個根, 因爲我們知道兩根和是8 , 就可以把另外一個根算出來等於$3 + 8 = B$, 因此兩根是3 和B, 故共有2個錯誤, 分別是第3個 bit 和第11個 bit, 這樣就可以把錯改回來。

在做 DEC BCH code 解碼的時候, 因爲剛好它有很好的關係, 可以把$x + y$和xy算出來, 可是假設把這個碼擴展到不只改兩個錯, 可以改三個錯或者更多個錯, 那時候就不能很簡單地算出來, 它有一個特別的 algorithm 在算, 這個 decoding algorithm很有名, 叫做 Berlekamp–Massey algorithm, 發明的兩個人原都是 Shannon 在 MIT 的學生, 另外有幾位日本人也提出可用 Euclid′s algorithm 來解碼。

剛才提到把這個碼擴展到不只可以改兩個錯, 那麼要如何去做? 前面改兩錯的時候是在α底下再加α^3, 那改三個錯怎麼辦? 要再加什麼? 實際上只要再加α^5就可以改三個錯, 再加α^7 就可以改四個錯, 這樣一直下去, 可以證明, 如果要改t個錯, 一直加到α^{2t-1}就可以了, 當然你的 columns 要夠多才行。基本上這個碼的矩陣就是$\alpha, \alpha^3, \alpha^5, \cdots$,這樣一直排列下去, 這就是所謂一般情形的 BCH code。

四.Reed-Solomon Codes

現在要介紹的這個碼是很有名的, 同時也可能是賺錢賺最多的一個碼, 比方說雷射唱盤裡面用的就是這個碼, 這個碼是由兩個人所發明的, 一個是 Reed, 另一個是 Solomon, 所以稱爲 Reed–Solomon code。這個碼是 nonbinary code, 是在1960年由 Reed 和 Solomon 兩個人提出來, 當時他們是在 MIT 的 Lincoln Laboratory, 這篇論文只寫了兩頁, 這兩頁就把碼提出來, 而解碼的方法是後來別人做的。

基本上, 這個碼是(n, k) code, 它的 minimum distance 剛剛好是$n - k + 1$, 然後這個碼不是 binary 的, 先前我們看的幾個碼, 它們的每個 bit 都是 binary, 而這個碼

它本身的 symbol 就是在一個 finite field 裡面的數, 通常在編碼的時候, 我們是把每m個 bits(如圖6) 視爲在 finite field $GF(2^m)$裡面的一個數, 然後拿它來運算。

$$\underbrace{10010101}_{m\text{ bits}}\ \underbrace{01010011}_{m\text{ bits}}\ \cdots\cdots$$

圖 6

那它的原理是怎麼樣? (見圖7) 基本上把直線上的點想像是在$GF(2^m)$ 裡面的一個數, 不是原來的實數, 這是爲了要畫圖, 所以要這樣想像, 不然 finite field 裡面的數就沒辦法畫了。假設要傳兩個數要怎麼傳? Reed 與 Solomon 基本上的想法是這樣: 若有兩個數, 則將這兩個數用平面上的點來表示, 橫座標表示1, 2, 等距的, 而縱座標則表示這兩個數在數線上的位置, 因兩點可以決定出一條直線出來, 當要傳這兩個數時, 就把這條直線畫出來, 若要改兩個錯, 就在後面加四個點上去 (如圖8), 因爲如果要改兩個錯, 它的 minimum distance 至少要5, 而剛才說這個碼的 minimum distance 是$n-k+1$, 因此$n-k+1=5, n-k=4, n-k$就是新加的個數, 所以要改兩個錯的時候, 就把直線找出來, 然後在3, 4, 5, 6的地方各取出一個值, 再連同原來的兩個數, 一共六個依序一起傳過去。假如收到的時候有錯, 錯成像圖9的樣子, 這六個點不在同一直線上, 可是我們知道它們原來應該在一條直線上, 所以就去找一條直線, 使得這條線上有最多的點, 很明顯的是如圖10所畫的這條線, 因此, 我們知道是第二點和第四點錯了, 就把這兩點拉回來, 這樣就可以把錯改回來, 基本上, 這個碼的觀念就是這樣。但如果錯三個點可能就會不對了, 因爲可能錯的那三點也是在一條直線上, 這樣我們可能就會找到不對的直線, 因此像前面的做法只能改兩個錯。

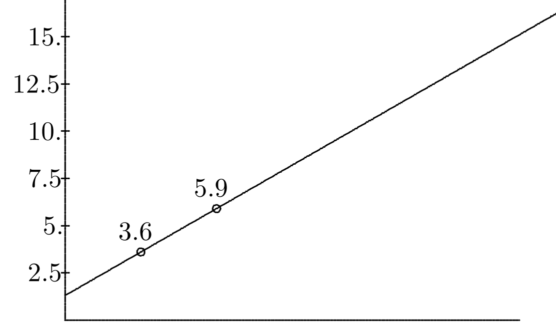

圖 7

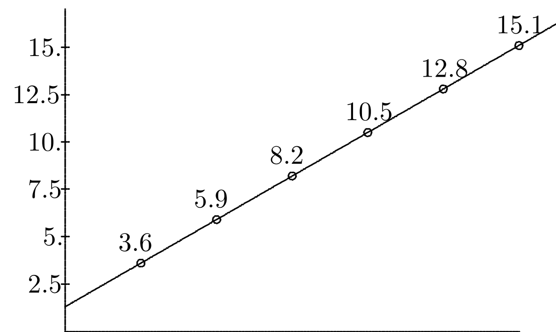

圖 8

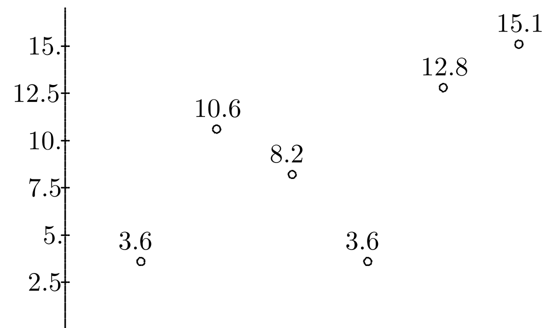

圖 9

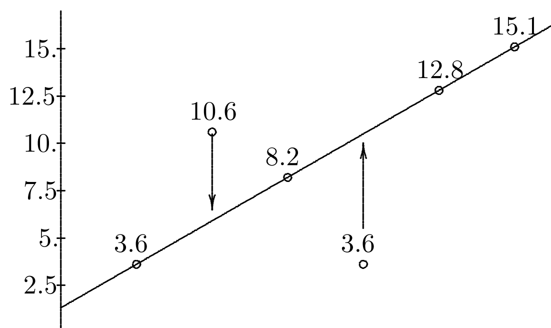

圖 10

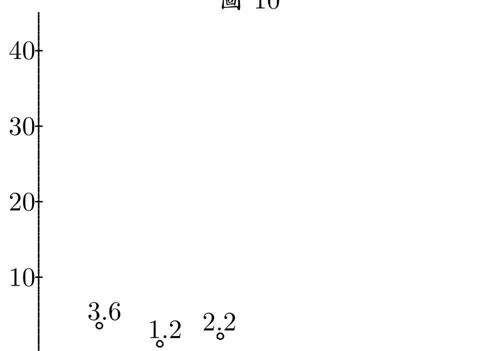

圖 11

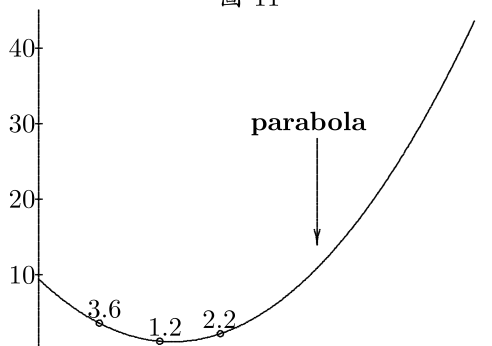

圖 12

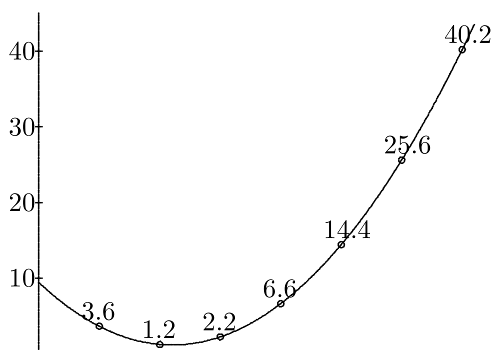

圖 13

剛才是假設要傳兩個 data, 兩個m bits, 假設現在要傳三個 (如圖11), 剛剛兩點可以畫一條直線, 那三個點我們可以畫一條二次曲線 (拋物線)(如圖12), 如果我們還是要改兩個錯, 就在上面再加四點 (如圖13), 每多加兩個點就可以多改一個錯, 改的方法跟兩個 data 的情形類似, 我們找出一條有最多點在上面的拋物線, 然後把不在線上的點拉回來。傳多個 data, 改多個錯的時候, 也是同樣的道理, 如果要傳ℓ個點, 就找一個次數是$\ell-1$的多項式, 因爲ℓ 點可以決定一個$\ell-1$次的多項式, 然後看要改幾個錯, 就多加兩倍的數, 譬如改t個錯, 就多加$2t$個數。所以它的觀念就是這麼簡單, 只是這些都是 finite field裡的數, 它們的加減乘除也是 finite field 裡的加減乘除, 因此, 根本畫不出圖來, 並不是像剛剛那些圖上的東西, 那樣畫只是讓我們有一個想像的空間可以去理解它。

五. Compact Disk Digital Audio System

最後稍微介紹一下 Compact Disk Digital Audio System, 存在這上面的資料是數位 (digital) 的, 原來是聲音或音樂, 經過取樣、量化之後, 再變成二進位, 變成0與1, 所以是數位的, 不是像傳統的唱盤、錄音帶是類比 (analog) 的訊號。disk 的直徑是12公分, 兩面裡只有反面存資料, 全部可以存74分鐘, 資料速率是1.5×10^6 bits/sec, 全部74分鐘差不多是6×10^9 bits 在上面, 這些是資料部分, 多加的部分並不算在裡面。另

外, 這上面有一個 track 在繞, 如果把它拉直的話, 全部大概有3.5英哩長, 每個 track 的寬度差不多是0.5×10^{-6}公尺, 大概是綠色光的波長, 因此看起來反光有點綠綠的。它所用的碼我們叫做Cross–Interleaved Reed–Solomon Codes, 平常簡寫成 CIRC, 它包括兩個 Reed–Solomon Codes, 一個是 (32, 28), 一個是 (28, 24), 每一個 symbol 都是8 bits, 第一個碼是 (32, 28), 所以 redundancy 是4, 可以改兩個錯, 另一個 (28, 24) 碼也是一樣可以改兩個錯, 然後用一種特殊的方式把這兩個碼結合在一起, 我們稱做 cross interleaving。那麼在這全部裡面有多少比例是眞正的資料? 32 bits 裡面有28 bits 資料, 28裡面有24 bits 資料, 所以全部比例是$\frac{28}{32} \cdot \frac{24}{28} = \frac{3}{4}$, 其中$\frac{3}{4}$的比例是資料, 前面提到 CD 上面有$6 \times 10^9$ bits 是資料, 因此有2×10^9 bits 是編碼之後加進去的。這樣的碼大概連續錯差不多4000個 bits 都可以改回來, 也就是差不多2.5mm track, 假如是錯到8mm track, 它也可以偵測 (detect) 出來, 所以通常錯不太多的時候, 它會直接改回來, 如果錯多的話, 它可以偵測到有錯, 然後用內插的方式把中間的音樂值算出來, 差別是耳朵聽不出來的, 因此在 CD 片上面刮一道長度小於8mm 的刮痕, 基本上不太會有問題。通常 CD 片上面另可能會有灰塵或指紋等等東西, 它就需要有錯誤更正碼來保護它, 假使錯的程度還在它的錯誤更正能力範圍內, 基本上就跟沒有錯是一樣的, 這是錯誤更正碼在日常生活中一個非常實際的應用。

參考資料

1. E. R. Berlekamp, ed., *Key Papers in the Development of Coding Theory.* New York: IEEE Press, 1974.
2. R. E. Blahut, *Theory and Practice of Error-Control Codes.* Reading, MA: Addison-Wesley, 1983.
3. S. Lin and D. J. Costello, Jr., *Error Control Coding, Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1983.
4. R. J. McEliece, *The Theory of Information and Coding.* Reading, MA: Addison-Wesley, 1977.
5. R. J. McEliece, *Finite Fields for Computer Scientists and Engineers.* Boston, MA: Kluwer Academic Publishers, 1987.

—本文作者任教於清華大學電機系—