南　　華　　大　　學

資訊管理學系

碩士論文

快速基因完全比對演算法

A fast exact gene matching algorithm

研 究 生：孫顯智

指導教授：廖怡欽

中華民國　102　年　7　月　21　日

誌　　謝

在這篇致謝經過多次刪減及修改後，最後還是決定以直白且不經修飾的詞語來撰寫這篇致謝。在此必須先說明，雖然這篇致謝的用字遣詞以及在語句通順上，跟參考過後的眾多致謝文相比後明顯遜色，但這篇致謝文中卻包含著我滿滿的感謝。

首先，這篇論文得以順利完成，其中最大的功臣就是我的指導老師廖怡欽教授。人說羅馬不是一天造成的，對我來說論文的完成也不是兩年就可以造成的。要是沒有廖老師在大學時代為程式語言及演算法等專業科目奠定下基礎；要是沒有廖老師在大學時代專題的細心指導；要是沒有廖老師在研究所時苦口婆心地指導，我想也不會有這篇論文的產生。從大學的專題一直到碩士畢業這段期間，廖老師總是對我抱持著很大的期望，當然，有期望就會有失望，而且這一路上還伴隨著很多失望。儘管如此，老師還是沒有放棄我。在研究方面，老師在百忙之中還要花費許多時間指導我程式正確的撰寫方式以及協助我的論文修正；在生活方面，老師總是藉由每次開會，指導我在生活上應該注意的小細節以及要時常在乎他人的感受；在面對未來的職場方面，老師總是告訴我許多以後該注意的事項。簡單來說，由於跟隨著廖老師的腳步，我在這幾年來真的受益良多，非常感謝。

此外，還要感謝這幾年來陪伴我的南華大學資工系及資管系。感謝系上老師對學生的細心指導，而且非常關心學生的狀況，不論是在課業上或是生活上；感謝助理(珍玉姊、伊姊、佩佩)總是能把我當朋友一樣看待，並且不厭其煩的幫助我解決許多問題；感謝碩士班同學級學長(小

偉、阿彥、育慧、于斌、漢宇、小紅、熊、PK、范范、默默、昱翔、久慧、熊董、老三、峰熙、黃西瓜等人)幫助我解決在課業上或是生活的總總疑難雜症，且讓我感到碩士生活並不寂寞無聊。另外，還要感謝資訊室中的各位成員(壽哥、隆哥、全福大哥、淳淳、麗秋)，不但讓我在這幾年來學到許多電腦維修及管理相關技術，還讓我可以自己賺取生活所需的費用。

當然還要感謝我的家人，包括哥哥、姊姊、媽媽。雖然不常見面，但總是能適時地對我伸出援手，關心我生活上的需求，讓我可以順利完成論文並且畢業。

最後，我很感謝我可以就讀南華大學，讓我遇見這麼多好的老師、助理以及同學，讓我這幾年的生命沒有留白，實在是萬分感謝!

摘　　要

隨著基因定序成本的降低，取得基因序列變得越來越容易，透過比對基因序列與基因片段，可達到身分識別、親屬關係鑑定、疾病預防及診斷等應用。現有字串比對演算法，雖可進行基因比對，但比對速度緩慢。為了提升基因比對速度，Srikantha 等人於 2010 年提出一套快速基因完全比對演算法。該演算法使用下採樣與雜湊表技術，可有效降低基因比對的時間複雜度。該演算法雖可降低時間複雜度，但當基因片段長度不足時無法使用，且存在許多無效的運算動作。為了提高該演算法的可用性以及提高基因比對速度，本論文提出三個改善方法。其中『多連續位置清單擷取方法』用來改善該演算法在基因片段長度不足時無法順利執行的情況；『線性位置過濾方法』及『去除無效的位置過濾動作』用來降低基因比對的時間複雜度。實驗結果顯示，所提方法在基因長度不足時仍可有效使用雜湊表內容，達到提升基因比對速度的效果，有效提升演算法的可用性。在一般情況下所提方法也可有效減少 38%~95%的比對時間。

關鍵詞：完全基因序列比對、字串比對

Abstract

With the decreasing of the DNA sequencing cost, to obtain the DNA sequence of a person becomes easier than before. Having a DNA sequence, we can check if a specific gene segment appears in it for purposes of identity recognition, paternity testing, and disease diagnosis and prevention. Existent string matching algorithms can be easily applied on such problems (gene matching problems) without any modification, but always takes a lot of computational time. To increase the gene matching speed, Srikantha et al. proposed a fast exact gene matching algorithm in 2010 using the down-sampling and hash table techniques. Srikantha's algorithm can effectively reduce the time-complexity of the gene matching process, but cannot be used for short gene segments and contains many redundant operations. To increase the availability of the algorithm and the gene matching speed, this thesis presents three improving methods. Where the multiple continuous location-lists retrieving method is used to make the algorithm applicable for all lengths of gene segments. The linear location filtering and the redundant filtering operation removing methods are used to reduce the time-complexity of gene matching process. Experimental results reveal that the proposed algorithm can effectively utilize the information in hash table to improve the gene matching speed for all lengths of gene segments. In general, the proposed algorithm can effectively reduce about 38% to 95% computational time.

Keyword : Exact gene sequence matching, String matching.

目錄

誌　　謝 I

摘　　要 III

Abstract IV

目錄 V

表目錄 VII

圖目錄 VIII

第一章、　緒論 1

1.1. 研究背景 1

1.2. 研究動機 3

1.3. 研究目的 8

1.4. 研究架構 8

第二章、　完全 DNA 比對演算法 9

2.1. Knuth Morris Pratt (KMP) 9

2.2. Karp Rabin (KR) 15

2.3. Srikantha 方法 18

第三章、　快速比對方法 27

3.1. 多連續位置清單擷取方法 27

3.2. 線性位置過濾方法 28

3.3. 去除無效的位置過濾動作 30
3.4. 快速比對演算法 31
第四章、 實驗結果 36
4.1. 實驗環境 36
4.2. 多連續位置清單擷取方法執行效能分析 37
4.3. 線性位置過濾方法執行效能分析 39
4.4. 去除無效的位置過濾動作執行效能分析 42
4.5. 快速基因比對演算法分析 45
第五章、 結論 47
參考文獻 49

表目錄

表 1、常見快速字串比對演算法 5

表 2、雜湊表內容 20

表 3、多連續位置清單擷取方法比對所需時間(ms) 37

表 4、字串比對演算法比對所需時間(ms) 38

表 5、DSSAHA 演算法比對所需時間(ms) 39

表 6、使用線性位置過濾方法改良後比對所需時間(ms) 40

表 7、線性位置過濾方法可以降低所需時間之百分比 40

表 8、去除無效的位置清單過濾動作改良後比對所需時間(ms) 42

表 9、去除無效的位置清單過濾動作可以降低所需時間之百分比 43

表 10、所提演算法之比對所需時間(ms) 45

表 11、所提方法可以降低所需時間之百分比 46

圖目錄

圖 1、Full Search 演算法流程圖 4

圖 2、KMP 演算法流程圖 13

圖 3、KR 演算法流程圖 16

圖 4、DSSAHA 完整演算法流程圖 26

圖 5、多連續位置清單擷取方法範例 28

圖 6、本論文所提之演算法完整流程圖 35

第一章、 緒論

1.1. 研究背景

人體的性狀表現及遺傳因子決定於細胞核中 23 對染色體內的去氧核醣核酸(Deoxyribonucleic acid, DNA)，DNA 的組成是由四種不同的鹼基(分別由 A、T、C、G 代表)隨機串連而成[1]，透過 DNA 定序技術[2]可取得 DNA 的鹼基排列方式。定序完成後的 DNA 可表示成一串 DNA 序列，例如一個依序由鹼基 A、T、G、C、C、A、T、C、T、C 所組成的 DNA 序列，可表示成" ATGCCATCTC"序列。人體的遺傳基因則由 DNA 序列中特定的片段決定，也就是說如果有一段對應特定遺傳基因的 DNA 片段出現在某人的 DNA 序列中，則該人便具有該遺傳基因。

人類基因組計畫（Human Genome Project, HGP）[3]起始於 1990 年，主要宗旨在測定組成人類染色體中所包含的 30 億個鹼基對組成的 DNA 序列，進而繪製人類基因組圖譜，破譯人類遺傳信息。透過比對個人 DNA 與疾病基因，可得知該人是否為某些疾病的高危險群，而可提早採取適當的預防措施與醫療行為，降低得病的機率及醫療成本，例如安潔莉娜裘莉檢驗出其罹患乳癌機率較一般人高，因此先行將乳房切除以免罹癌。人類基因組計畫原先預計於 2005 年完成，由於科技的進步，提早於 2003 年 4 月份正式完成計畫。雖目前已可破譯個人 DNA 序列，但要取得個人

DNA 序列(DNA 定序)仍不容易，需花費大量金錢與時間成本。

為了降低 DNA 定序所需的成本，目前已有許多研究團隊投入研發快速且便宜的 DNA 定序技術，美國國家衛生研究院[4]與美國 XPrize 基金會[5]更提供一千萬美元的鉅額獎金舉辦 DNA 定序比賽，希望在 2014 年以前將個人 DNA 定序成本降到 1000 美元以下。目前已有許多公司[6][7][8][9]提供 DNA 定序服務，而且已經可以將個人 DNA 定序的售價降至 3,000 美元，定序時間降至 24 小時以下。Jonathan 等人則提出一個可將定序成本降至 1,000 美元的方法[10]，Jonathan 等人更於 2012 年 7 月報名參加DNA定序比賽，成為首位報名者。可預期未來隨著科技的進步，取得個人 DNA 序列的時間及金錢成本將會持續降低。

完成基因定序後，可藉由 DNA 序列比對方法，檢驗個體中是否包含特定 DNA 片段。目前 DNA 的應用中，包括人身鑑定[11]、親屬關係鑑定[12]、部分單一基因疾病診斷[13]都可透過短縱列重複序列(Short Tandem Repeat, STR)的檢驗來完成。STR 是由 2~8 個核甘酸所組成，且片段連續重複出現，舉例來說，假設有一(CAG)7，即代表"CAG"連續重複出現 7 次構成"CAGCAGCAGCAGCAGCAGCAG "片段。有些疾病的產生就是 STR 過度擴增所造成，例如遺傳性智障 Fragile X 症，其病因是特定位置的STR出現大量的擴增(n<50 為正常、50<n<200 為帶因的前突變、

n>200 為患者)，因此我們可以透過片段比對的方式得知人體中 STR 的分佈狀況，是否有在部分區域 STR 大量擴張造成疾病的產生。

1.2. 研究動機

電腦具有高運算效能、容易存取傳輸、可重覆計算等優點，是相當重要的 DNA 比對工具，目前國內外有許多研究團隊從事相關研究[14][15][16][17]。使用電腦進行 DNA 比對，基本上是將每個鹼基當成一個字元，組合而成的 DNA 序列則當成一個字串，然後使用一般的字串比對方法進行比對。常見的比對方式又可分為容錯比對與完全比對兩種。容錯比對的做法是透過 Levenshtein Distance[18]概念，經由插入(insertions)、刪除(deletions)、置換(substitutions)編輯過程，在個人 DNA 序列中找尋與目標基因相似度最高的片段位置；完全比對的做法是透過比對兩片段中每個相對位置的字元是否相同，在個人 DNA 序列中找尋是否有與目標基因完全相同的 DNA 片段存在。容錯比對與完全比對應用於基因比對中各有其貢獻，容錯比對可應用於物種間相似度及部份疾病診斷等領域，完全比對則可應用於 STR 部分特定應用領域方面。在完全比對的應用中，可提供較為快速的資料比對，而後也可衍生至一般的字串比對中使用，故本論文僅針對完全比對進行探討。

在現有的完全比對方法中 Full Search 是最直覺的方法，假設個人

DNA 序列為 S， DNA 樣本為 P(例如：疾病基因片段)，L_S 表示 S 序列的長度，L_P 表示片段 P 的長度，Full Search 的做法是將 S 中的 0 到 L_S-L_P+1 中所有的位置當做是開頭，取 L_P 長度的片段與 P 進行比較是否相同。Full Search 的比對過程如圖 1 所示：

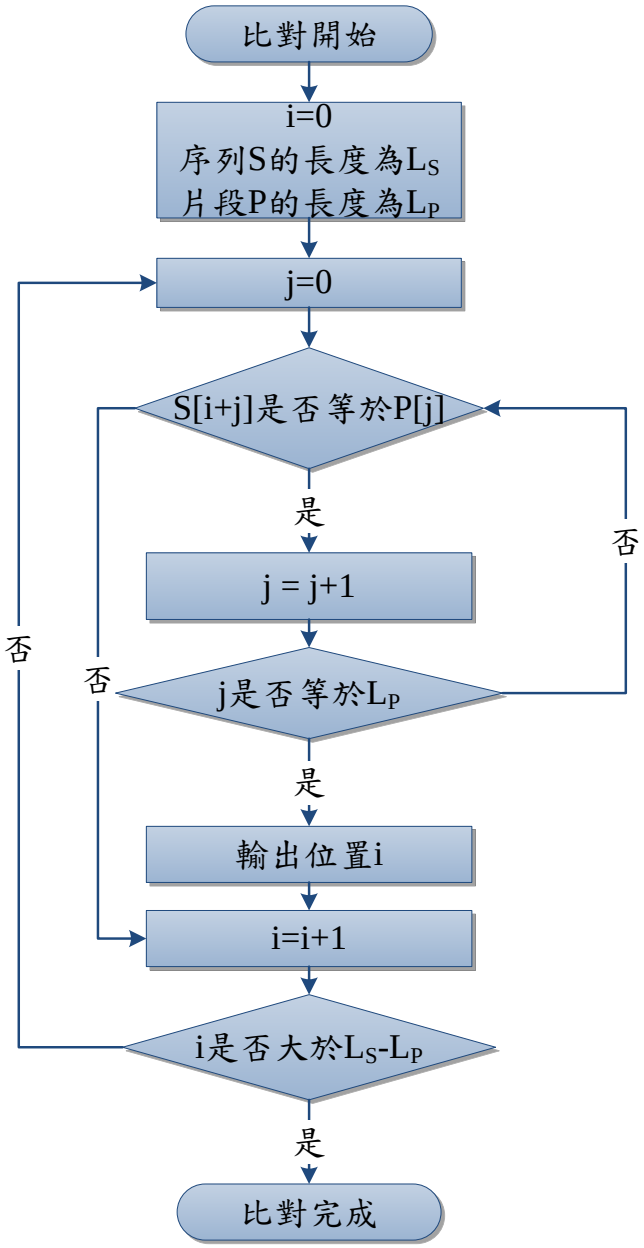

圖 1、Full Search 演算法流程圖

Full Search 的演算法如下所示：

Full Search 演算法
1. 令 i = 0
2. 令 j = 0
3. 若(S[i+j]≠P[j]), 到步驟(6)
4. j=j+1
5. 若(j = L_P)，輸出位置 i，並到步驟(6) 否則到步驟(3)
6. i=i+1
7. 若(i ≤L_S-L_P)，跳到步驟(2) 否則代表比對完成

雖然 Full Search 方法相當簡單，但時間複雜度高，為了降低比對時間，有許多學者提出不同的快速字串比對演算法[19][20][21][22][23][24]。表 1 所列為部分常見的快速字串比對演算法及其空間與時間複雜度：

表 1、常見快速字串比對演算法

演算法	前處理時間複雜度	空間複雜度	字串比對時間複雜度
Boyer Moore[19]	O(L_P)	O(L_P)	O(L_S×L_P)
Knuth Morris Pratt[20]	O(L_P)	O(L_P)	O(L_S+L_P)
Horspool[21]	O(L_P)	O(1)	O(L_S×L_P)
Karp Rabin[22]	O(L_P)	O(1)	O(L_S+L_P)
Shift Or[23]	O(L_P)	O(L_P)	O(L_S)
Suffix Tree Based[24]	O(L_S)	O(L_S)	O(L_P)

表 1 所列之資訊僅可做為挑選演算法時的參考，實際應用時，會因資料內容的不同而有不同的效果。例如 Suffix Tree Based 方法具有較佳的比對時間複雜度 $O(L_P)$，但並不適合應用於 DNA 比對，主要原因是因為其前處理及空間複雜度與 L_S 成正比。人體 DNA 內含三十多億個鹼基，因此 Suffix Tree Based 需要大量的記憶體及較多的前處理時間，記憶體需求越高也會降低其執行的效率。Kunth Morris Pratt(KMP)演算法採用類似 Full Search 的比對方式，但每次比對失敗後，透過前處理所建立的失誤函數可以省略部分已比對過的字元。但 DNA 片段中排序不具有規則，隨機性高，因此 KMP 無法建立出一個效益較高的失誤函數，導致在執行效率上受到限制。Shift Or 雖然時間複雜度只有 $O(L_S)$，但由於基因資料 $L_S>>L_P$ 的資料特性，使其比對時間並不理想。

為了提升 DNA 比對的效能，後來又有學者針對 DNA 資料而設計適合 DNA 比對的快速演算法[15][25]。其中 Sequence Search and Alignment by Hashing Algorithm[25] (以下簡稱 SSAHA)是一個相當有效的 DNA 比對方法。SSAHA 的作法是先將 DNA 序列 S 切割成 L_S-Q+1 個可重疊且長度為 Q 的 DNA 片段(稱為 Q-gram)，以每個 Q-gram 當作索引值建立一個雜湊表(Hash Table)，並於雜湊表中記錄每個 Q-gram 的起始位置及其所在的序列位置。完成雜湊表的建立後，接著就進行字串比對的動作，做法是

先將樣本 P 切成多個不重疊的 Q-gram，然後依序使用 Q-gram 做為雜湊表的索引，取出 S 中與 Q-gram 相同片段出現過的位置清單，如果取出的值都在同一序列，且連續出現$\frac{L_P}{Q}$個起始位置差異為 Q 的位置，即代表其可能為片段 P 出現於 S 的位置，再使用 Full Search 進行確認即可。SSAHA 的空間複雜度與前處理時間為 $O(L_S)$，時間複雜度為 $O(\frac{L_S}{4^Q}\log(\frac{L_S}{4^Q}))$，相對於表 1 的字串比對方法，大幅的提昇了比對的效能。

SSAHA 雖大幅度的加速了比對的速度，但其空間複雜度過高，為了降低 SSAHA 的空間複雜度及提升比對速度，Srikantha 等人於 2010 年提出了一套快速基因比對方法[15](以下簡稱 Srikantha 方法)。Srikantha 方法是一個相當有效率的快速 DNA 片段完全比對方法，其作法是先為 DNA 序列進行 M 下採樣後再建立雜湊表；雜湊表的鍵值由長度 Q 的 DNA 片段產生，雜湊表內的每個項目(entry)包含一串可能的位置，該些位置代表雜湊表鍵值出現在 M 下採樣後之 DNA 序列的位置。DNA 片段比對時，先將 DNA 片段進行 M 下採樣並切割成多個長度為 Q 且不重疊的 DNA 片段後；再由雜湊表中取出該些 DNA 片段可能出現在下採樣後之 DNA 序列的位置清單，經交叉過濾所有位置清單的內容後便可找出可能的位置，最後再使用完全比對方法檢查這些位置，確認其正確性。該研究大幅提升 SSAHA 的比對速度，將時間複雜度降為 $O(\frac{L_S}{4^Q}\log(\frac{L_S}{M4^Q}))$，也將空

間複雜度降為 $O(\frac{L_S}{M})$。

Srikantha 的方法是目前已知最快速的完全 DNA 比對演算法，但設計上仍有改善空間。經過詳細分析該演算法的作法後，發現當 L_P 小於 Q×M 時，會造成無法產生足夠長度鍵值的情況，導致程式無法順利執行。因此本論文設計了一個『多連續位置清單擷取方法』用以解決上述的情況。此外本論文也發現該演算法的主要執行時間花費在交叉過濾所有位置清單的內容上，為了有效降低位置過濾所花的時間，本論文提出兩個有效的改善方法，分別為『線性位置過濾方法』與『去除無效的位置過濾動作』來降低過濾時間。所提方法可有效將時間複雜度由 $O(\frac{L_S}{4^Q}\log(\frac{L_S}{M4^Q}))$降至$O(\frac{L_S}{4^Q})$。

1.3. 研究目的

本論文主要目的在改善現有 Srikantha 方法，提出一個高可用性與高效能的快速 DNA 完全比對方法。

1.4. 研究架構

本論文第 2 章將介紹常用的字串比對方法以及 Srikantha 方法，第 3 章介紹本論文所提出來的改善方法，第 4 章提供實驗結果，第 5 章結論。

第二章、 完全 DNA 比對演算法

除了專門針對 DNA 比對所設計的方法外，傳統字串比對演算法也可直接用來解決完全 DNA 比對的問題，以下各節將介紹幾個常用的快速字串比對演算法以及 Srikantha DNA 比對演算法。

2.1. Knuth Morris Pratt (KMP)

假設要檢查 DNA 序列 S 中是否出現片段 P，KMP 演算法的做法是先用片段 P 建立失誤函數，在比對過程中遇到比對失敗的情況時，可藉由失誤函數省略掉部分已比對過的字元，加速比對速度，失誤函數及比對方法分別說明如下：

2.1.1. 失誤函數

失誤函數的做法是先建立一個失誤函數陣列 D 存放失誤值，D 的陣列長度為 L_D，且 L_D 等於 L_P(片段 P 的長度)。令 D[k]代表陣列 D 中的第 k 個失誤值，D[0]的初始值為-1，D[1]~D[L_P-1]則是透過比對 P[i]與 P[j]是否相同或是連續相同來決定其內容值，i 值的範圍為$0 \leq i < L_P$，j 值的範圍為$1 \leq j < L_P$。建立失誤函數的詳細演算法如下：

失誤函數建立演算法

1. 令 D[0]=-1
2. 令 i=0 , j=1
3. i = D[j-1]
4. 若 P[j]等於 P[i+1]且 i 小於 0，並到步驟(5)

   否則 i=D[i]，並回到步驟(4)
5. 若 P[j]等於 P[i+1]，則 D[j] = i+1

   否則 D[j]=-1
6. 若 $j<L_P$，則 j++,並回到步驟(3)
7. 失誤函數 D 建立完成

為了方便理解失誤函數的建立方法，以下舉一個簡單例子來說明，假設有一片段 P 為"ACTACTAACC"

1. i = D[0] = -1，j = 1

   因 P[0]不等於 P[1]，故 D[1]=-1
2. i = D[1] = -1，j = 2

   因 P[0]不等於 P[2]，故 D[2]=-1
3. i = D[2] = -1，j = 3

   因 P[0]等於 P[3]，故 D[3]=-1+1 = 0
4. i = D[3] = 0，j = 4

因 P[1]等於 P[4]，故 D[4]=0+1 = 1

5. i = D[4] = 1，j = 5

   因 P[2]等於 P[5]，故 D[5]=1+1 = 2

6. i = D[5] = 2，j = 6

   因 P[3]等於 P[6]，故 D[6]=0+1 = 3

7. i = D[6] = 3，j = 7

   因 P[4]不等於 P[7]，但 P[3]等於 P[7]，故 D[7]=0

8. i = D[7] = 0，j = 8

   因 P[1]等於 P[8]，故 D[8]=0+1=1

9. i = D[8] = 1，j = 9

   因 P[2]不等於 P[9]，故 D[9]=- 1

10. i = D[9] = -1，j = 10，因 j 等於 L_P，故比對完成

11. D={-1,-1,-1,0,1,2,3,0,1,-1}

2.1.2 字串比對

KMP 演算法在字串比對時，基本的做法與 Full Search 相同，但在遇到字元比對不相同時，會透過失誤函數中所提供的資訊，計算出可省略過部分已比對過的字元數不進行比較。假設兩字串比對到片段 P[j]時比對失敗，可透過失誤函數計算下個 j 的位置繼續比對，計算公式如下所示：

j = D[j-1]+1.....(1)

舉例來說，假設有一序列 S 為"ACTACTGACTACTAACC"，片段 P="ACTACTAACC"，當比對到 S[6]= 'G '時，發覺與 P[6]= 'A '不相同，則透過失誤函數計算片段 P 的下一個位置 j = D[6-1]+1=3，因此片段 P[3]可以直接對齊 S[6]再次進行比對，省去了 Full Search 比對 P[0]、P[1]、P[2]的時間。KMP 比對的流程如圖 2 所示：

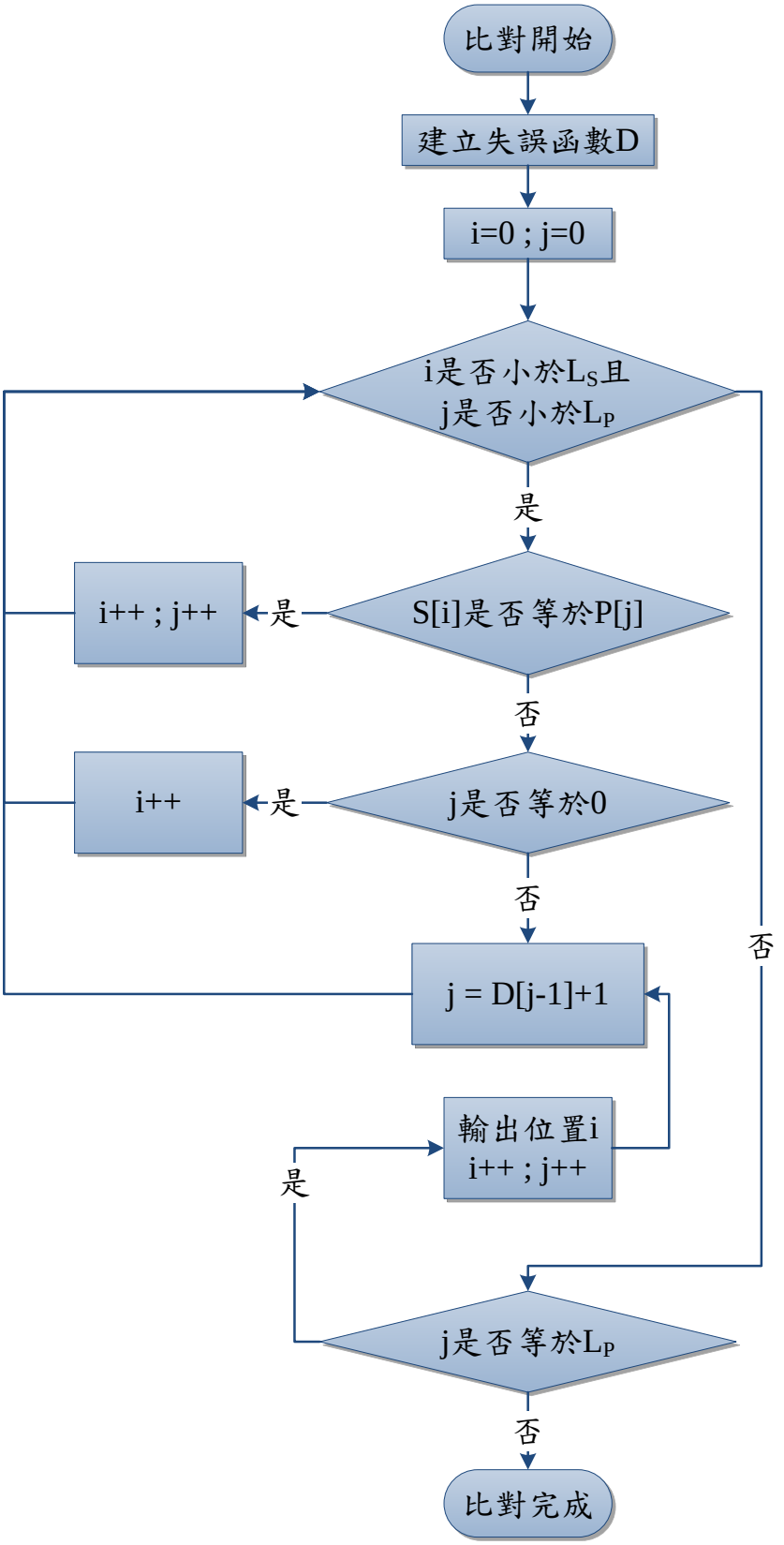

圖 2、KMP 演算法流程圖

KMP 完整的比對演算法如下所示：

KMP 字串比對演算法

1. 建立失誤函數 D
2. 令 i = 0 ; j=0
3. 若 i 小於 L_S 且 j 小於 L_P，到步驟(4)

   否則到步驟(7)
4. 若 S[i]等於 P[j]則 i++，j++，回到步驟(3)
5. 若 j 等於 0 則 i++回到步驟(3)
6. j = D[j-1]+1;回到步驟(3)
7. 若 j 等於 L_P 則輸出位置 i，i++，j++，並回到步驟(6)

   否則比對完成

2.2. Karp Rabin (KR)

KR 演算法的做法是透過比較片段 P 及從序列 S 中取出之長度為 L_P 的片段 S_F，經過雜湊函式(Hash Function)轉換後的鍵值是否相等，來判定是否可能相同，如果鍵值相同，再利用 Full Search 檢查，如果不相同則 P 與 S_F 不可能相同，可以不需進行完整比對，有效的減少比對次數。KR 演算法將比對過程分為前處理及字串比對兩部分，詳細的前處理步驟及字串比對方法分別於下列兩個小節介紹。

2.2.1 前處理

KR 演算法在前處理的過程中需先建立位移運算元 d，建立的方法是將 d 設定為 1 後，再將 d 向左位移 L_P-1 次即建立完成。d 主要的用途是當字串比對失敗後，可以將欲刪除的字元與 d 相乘後從鍵值中減去，並加上 S 中的下一個字元後便可產生新的鍵值。

產生 d 值後,再將片段 P 與 S 的子字串 S[head]~S[head+L_P-1](head=0)使用雜湊函式轉換出鍵值 h_P 及 h_S 後，即完成前處理的過程。

2.2.2 字串比對

字串比對的方式，是透過比對 h_P 及 h_S 兩數值是否相等來判斷，當每次比對失敗時，就將 h_S 的值減去 S[head]乘上 d，再將 h_S 向左位移後加上 S[head +L_P]所代表的數值即完成新 h_S 建立，h_S 的計算公式如下所示：

$h_S=(h_S-S[head]*d)<<1+S[head+L_P]$.....(2)

KR 字串比對流程如圖 3 所示：

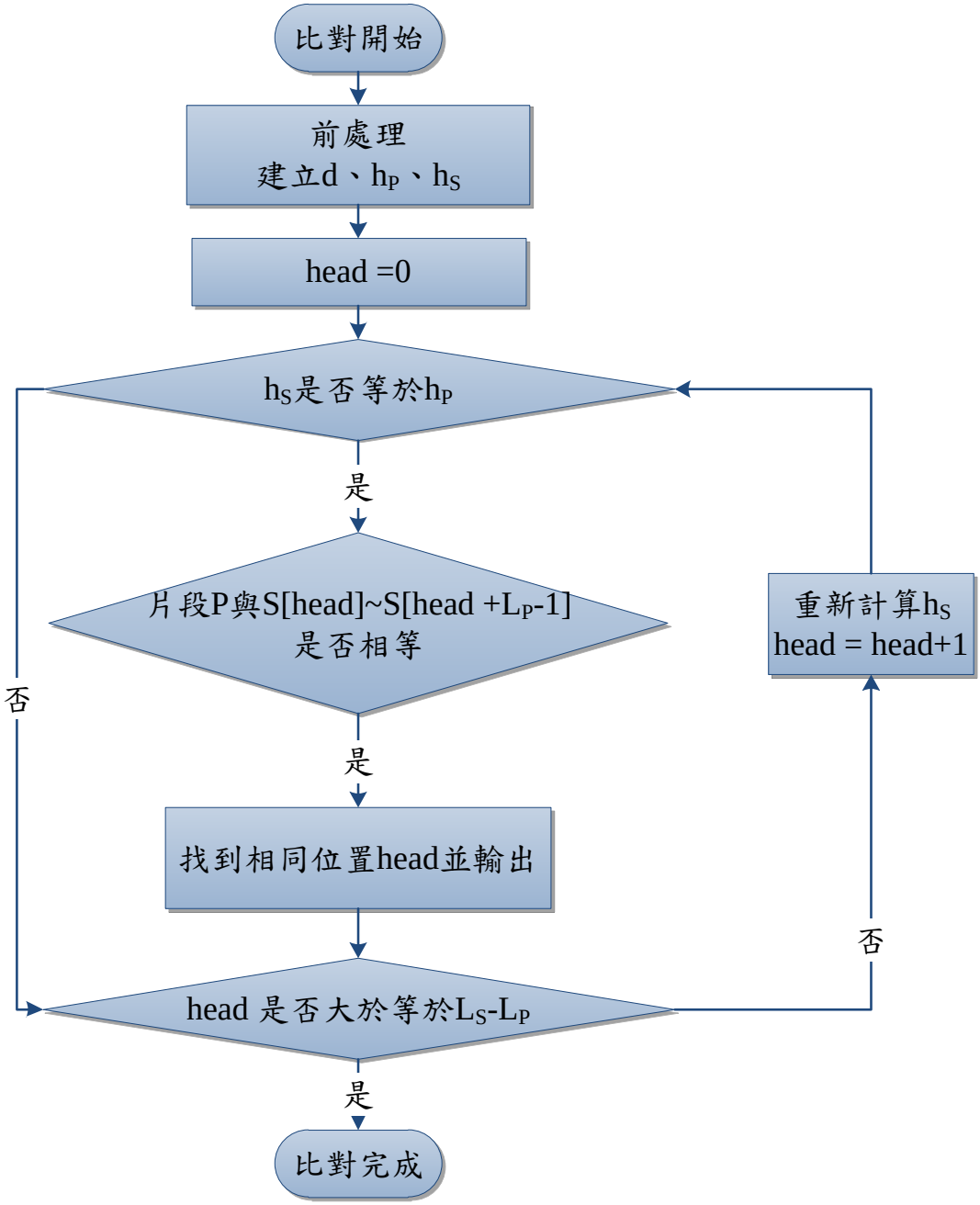

圖 3、KR 演算法流程圖

KR 完整字串比對演算法如下所示：

KR 字串比對演算法

1. 透過前處理建立 d、h_P、h_S
2. head =0
3. 若 h_S 不等於 h_P 則到步驟(5)
4. 若片段 P 與 S[head]~S[head +L_P-1]使用 Full search 比對相等，則找到相同位置 head 並輸出
5. 若 head >=L_S-L_P，則代表比對完成

   否則 h_S=(h_S-S[head]*d)<<1+S[head +L_P]，head +=1 回到步驟(3)

為了方便理解 KR 演算法，以下舉一個簡單例子來說，假設有一序列 S 為"ACTACT"，片段 P 為"TACT"

1. 建立 d=8，h_P=84×8+65×4+67×2+84=1150

   h_S=65×8+67×4+84×2+65=1021
2. head =0
3. 由於 h_P≠h_S

   h_S=(1021−65×8)<<1+67=1069，head = head +1=1
4. 由於 h_P≠h_S
5. h_S=(1069−67×8)<<1+84=1150，head = head +1=2
6. 由於 h_P=h_S，找到相同位置 2，i>=L_S-L_P，比對完成。

2.3. Srikantha 方法

Srikantha 方法包含兩個階段，第一階段是使用 DNA 序列建立雜湊表，供第二階段比對 DNA 片段時使用；第二階段則是使用第一個階段所產生的雜湊表進行 DNA 比對，這兩個階段的作法分別說明如下。

2.3.1. 雜湊表建構階段

為了減少雜湊表的儲存空間，在 Srikantha 的方法中，要為一個 DNA 序列建立一個雜湊表，先對 DNA 序列進行 M 下採樣，然後再使用下採樣後的 DNA 序列產生雜湊表。令 S_M 為一 DNA 序列 S 進行 M 下採樣後的結果，則 S_M 的長度 L_{S_M} 與 S 的長度 L_S 的關係如下：

$$L_{S_M}=\left\lceil \frac{L_S}{M} \right\rceil \quad (3)$$

S_M 的內容定義如下：

$$S_M[i]=S[i\times M]，\text{for } 0\le i< \left\lceil \frac{L_S}{M} \right\rceil \quad (4)$$

例如：M=3，S="ATCGGTAGTC"，則 S_3="AGAC"。

有了 S_M 後，接著再利用 S_M 產生 L_{S_M}-Q+1 個長度為 Q 的 DNA 片段(簡稱 Q-gram)。令 $S_{M,Q}=\{S_{M,Q}(i) \mid 0\le i\le L_{S_M}-Q\}$ 為 S_M 的所有 Q-gram 的集合，其中，$S_{M,Q}(i)$ 為利用 S_M 所產生的第 i 個 Q-gram，$S_{M,Q}(i)$ 的內容定義如下：

$$S_{M,Q}(i)[j]=S_M[i+j]，\text{for } 0 \le j < Q \quad (5)$$

例如：若 Q=4，S_M="ATCGGT"，則 $S_{M,4}=\{S_{M,4}(0), S_{M,4}(1), S_{M,4}(2)\}$；

$S_{M,4}(0)$="ATCG"，$S_{M,4}(1)$="TCGG"，$S_{M,4}(2)$="CGGT"。

有了 $S_{M,Q}$ 後，便可使用 $S_{M,Q}$ 來建立雜湊表，建立雜湊表的作法是利用 $S_{M,Q}(i)$的內容產生鍵值，然後再將 $S_{M,Q}(i)$在 S_M 內的起始位置(也就是 i)加入該鍵值所指到之雜湊表項目(entry)內的位置清單中。

鍵值的產生方式是將 Q-gram 中的每個鹼基轉換成相對應的二進位位元串列，一個鹼基對應到兩個 2 進位位元，其中，A、C、T、G 分別對應到 00_2、01_2、10_2、與 11_2。例如：若 Q-gram 為"ATGCC"，則轉換後的鍵值為 $0010110101_2=B5_{16}=181_{10}$。詳細的雜湊表建構演算法整理如下：

雜湊表建構演算法

---

1. 產生 S 的 M 下採樣結果 S_M
2. 產生 $S_{M,Q}=\{S_{M,Q}(i) \mid 0\leq i\leq L_{S_M}-Q\}$
3. 清除雜湊表內容
4. 令 i=0
5. 利用 $S_{M,Q}(i)$計算鍵值
6. 將位置 i 加入鍵值所指到之 entry 的位置清單
7. i=i+1
8. 若$i \leq L_{S_M} - Q$，跳到步驟(5)

為了方便了解雜湊表的建構方法，以下以一個簡單的例子來說明。

假設 M=3，Q=3

S="CATACATTTGACGGATACACATGTGACCAA"

則 S_3="CATGGTATGC"

$S_{3,3}$={$S_{3,3}$(0),$S_{3,3}$(1), …, $S_{3,3}$(7)}

={"CAT", "ATG", "TGG", "GGT", GTA", "TAT", "ATG","TGC"}

用 $S_{3,3}$ 建構而成的雜湊表內容如下表所示：

表 2、雜湊表內容

鍵值	**位置清單**
001011_2	1,6
010010_2	0
100010_2	5
101101_2	7
101111_2	2
111000_2	4
111110_2	3

有了雜湊表後，我們可以快速的為任何長度為 Q 的 DNA 片段找出其出現在 S_M 的位置。

2.3.2. DNA 片段比對階段

給定一 DNA 片段 P 以及一個 DNA 序列 S，要檢查 P 是否出現在 S 中，可先將 P 進行下採樣取得 P_M，再將 P_M 切割成多個長度為 Q 的 DNA 子片段，然後利用 S 的雜湊表找出 P_M 內各個子片段出現在 S_M 的位置清單，交叉比對各個子片段取出之位置清單過濾掉不可能的位置，將位置

清單中的位置轉換成未下採樣的位置，最後再使用 Full Search 方法檢查 DNA 序列 S 在該些位置是否真的出現 DNA 片段 P。以上過程詳細說明如下：

1. 對 P 進行 M 下採樣：下採樣後的長度$\mathbf{L_{P_M}}$與下採樣後的 DNA 序列 P_M 定義如下：

   $$\mathbf{L_{P_M}}=\left\lceil\frac{\mathbf{L_P}}{\mathbf{M}}\right\rceil \quad \text{.....(6)}$$

   $$P_M[i]=P[i\times M],\ \text{for } 0\le i<\left\lceil\frac{\mathbf{L_P}}{\mathbf{M}}\right\rceil \quad \text{.....(7)}$$

2. 將 P_M 分割成$\left\lfloor\frac{L_{P_M}}{Q}\right\rfloor$個長度為 Q 且不重疊的 DNA 片段(簡稱 NQ-gram)：令 $P_{M,NQ}=\{P_{M,NQ}(i) \mid 0\le i<\left\lfloor\frac{L_{P_M}}{Q}\right\rfloor\}$為 P_M 所有 NQ-gram 的集合，$P_{M,NQ}(i)$為利用 P_M 所產生的第 i 個 NQ-gram，其中，$P_{M,NQ}(i)$的內容定義如下：

   $$P_{M,NQ}(i)[j]=P_M[i\times Q+j],\ \text{for } 0\le j < Q \quad \text{.....(8)}$$

例如：若 Q=3，P_M="ACTGTAT"，則 $P_{M,NQ}=\{P_{M,N3}(0), P_{M,N3}(1)\}$，$P_{M,N3}(0)$="ACT"，$P_{M,N3}(1)$="GTA"。

3. 由雜湊表中取出位置清單：利用 $P_{M,NQ}$ 內的每個 NQ-gram 從雜湊表中取出位置清單。令 $H[P_{M,NQ}(i)]$為使用 $P_{M,NQ}(i)$產生之鍵值所取出的位置清單，$H[P_{M,NQ}(i)](k)$代表位置清單中的第 k 個位置，$L_{H[PM,NQ(i)]}$代表 $H[P_{M,NQ}(i)]$位置清單中的位置個數。例如：若 $P_{M,N3}(0)$ = " ATG "，鍵值為 001011_2；則使用該鍵值由表 2 的雜湊表中取出的位置清單為

$H[P_{M,NQ}(0)]=[1,6]$，其中，$H[P_{M,NQ}(0)](0)=1$，$H[P_{M,NQ}(0)](1)=6$。

4. 過濾不可能的位置：由於片段 $P_{M,NQ}(i)$在序列 P_M 中的位置為 $P_{M,NQ}(0)$的位置加上偏移量 $i\times Q$，因此若 P_M 片段完整的出現在 S_M 的第 X 個位置，則位置 $X+i\times Q$ 必定出現在 $H[P_{M,NQ}(i)]$中，而且針對每個可能的 i 都需符合。也就是 X 必須符合以下公式，才是合格的位置：

$$X+i\times Q\in H\,[P_{M,NQ}(i)],\ \text{for } 0\le i< \left\lfloor \frac{L_{P_M}}{Q} \right\rfloor \quad \text{.....(9)}$$

因此如果一個位置 X 出現在 $H[P_{M,NQ}(0)]$，但是其相對應的位置$(X+i\times Q)$沒有出現在 $H[P_{M,NQ}(i)]$，則可將 X 由 $H[P_{M,NQ}(0)]$中刪除。$H[P_{M,NQ}(0)]$是由雜湊表中取出的位置清單，刪除 $H[P_{M,NQ}(0)]$的位置不會改變雜湊表的內容。為了加快位置過濾過程，Srikantha 方法使用二元搜尋法(Binary Search)來檢查 $H[P_{M,NQ}(0)]$的相對應位置是否出現在後續的位置清單中$(H[P_{M,NQ}(i)]，1\le i< \left\lfloor \frac{L_{P_M}}{Q} \right\rfloor)$。

5. 轉換成相對應於 S 的位置：經上述步驟過濾剩下的可能位置為相對於 S_M 的位置，因此必須將 $H[P_{M,NQ}(0)]$內的位置轉換成相對應於 S 的位置後才能使用。令 Y 為 $H[P_{M,NQ}(0)](k)$相對應於 S 的位置，則 Y 的定義如下：

$$Y = H[P_{M,NQ}(0)](k)\times M \quad \text{.....(10)}$$

利用上述過程，雖可找到 P_M 出現在 S_M 的所有可能的位置。但無法找出 P 出現在 S 的所有可能位置，其原因為 M 下採樣後，若 P 出現在 S

的位置不是 M 的倍數，則無法利用 M 下採樣的結果找到位置。

為了解決這個問題，必須針對 P 做 0~M-1 個字元位移，然後針對不同位移後的內容各做一次前述 DNA 片段搜尋步驟，才能找出所有位置。

令 P^s 為 P 做 s 個字元位移後的結果，P^s 與 P^s 的長度 $\mathbf{L_{P^S}}$ 定義如下：

$P^s[i]=P[i+s]$，for $0 \le i < L_P-s$ .....(11)

$\mathbf{L_{P^S}}=L_P- s$.....(12)

由於加入新的變量 s，故公式(10)必須做適當的修正，修正後的公式分別如下：

$Y = H[\mathbf{P^S_{M,NQ}}(0)](k)\times M - s$.....(13)

詳細的 DNA 片段比對演算法整理如下：

DNA 片段比對演算法

1. 令 PP = Ø
2. 令 s=0
3. 產生 P^s
   - 3.1 產生 P^s 下採樣結果 $\mathbf{P^S_M}$
   - 3.2 產生 $\mathbf{P^S_{M,NQ}}$
   - 3.3 取出 $H[\mathbf{P^S_{M,NQ}}]$
   - 3.4 使用二元搜尋法刪除 $H[\mathbf{P^S_{M,NQ}}(\mathbf{0})]$ 中不符合公式(9)的位置
   - 3.5 使用公式(13)將 $H[\mathbf{P^S_{M,NQ}}(\mathbf{0})]$ 的位置轉換成相對應於 S 的位置後加入 PP
4. s=s+1

5. 若 s < M，跳到步驟(3)

6. 使用 Full Search 方法檢查 PP 中所有位置

為了方便了解序列比對的過程，以下使用一個簡單的例子來說明 DNA 片段搜尋的過程。假設 M=3，Q=3

P="TACATTTGACGGATACACATGTGACCA"

S=" CATACATTTGACGGATACACATGTGACCAA"

雜湊表與表一相同。則 DNA 片段比對過程如下：

1. PP= Ø

2. s=0

3. 產生 P^0=" TACATTTGACGGATACACATGTGACCA "

   3.1 產生$\mathbf{P_3^0}$="TATCACATC"

   3.2 產生$\mathbf{P_{3,N3}^0}$={$\mathbf{P_{3,N3}^0}$(0),$\mathbf{P_{3,N3}^0}$(1) ,$\mathbf{P_{3,N3}^0}$(2)}={"TAT","CAC","ATC"}

   3.3 H[$\mathbf{P_{3,N3}^0}$(0)] = {5}

   H[$\mathbf{P_{3,N3}^0}$(1)] = Ø ，H[$\mathbf{P_{3,N3}^0}$(2)] = Ø

   3.4 位置{5}不符合公式(9)故刪除 H[$\mathbf{P_{3,N3}^0}$(0)]= Ø

   3.5 由於 H[$\mathbf{P_{3,N3}^0}$(0)]為空集合，不進行位置轉換

4. s=s+1=1

5. s<M，跳到步驟(3)

6. 產生

P^1="ACATTTGACGGATACACATGTGACCA "

6.1 產生$\mathbf{P_3^1}$="ATGGTATGC"

6.2 產生$\mathbf{P_{3,N3}^1}$={$\mathbf{P_{3,N3}^1}$(0),$\mathbf{P_{3,N3}^1}$(1) ,$\mathbf{P_{3,N3}^1}$(2)}={" ATG","GTA","TGC"}

6.3 H[$\mathbf{P_{3,N3}^1}$(0)] = {1,6}，H[$\mathbf{P_{3,N3}^1}$(1)] = {4}，H[$\mathbf{P_{3,N3}^1}$(2)] = {7}

6.4 位置{6}不符合公式(9)故刪除 H[$\mathbf{P_{3,N3}^1}$(0)]= {1}

6.5 將位置{1}使用公式(13)轉成對於 S 的位置

(1×3–1 =2)加入 PP，PP={2}

7. s=s+1=2

8. s<M，跳到步驟(3)

9. 產生 P^2="CATTTGACGGATACACATGTGACCA "

9.1 產生$\mathbf{P_3^2}$="CTAGACGAA"

9.2 產生$\mathbf{P_{3,N3}^2}$={$\mathbf{P_{3,N3}^2}$(0),$\mathbf{P_{3,N3}^2}$(1) ,$\mathbf{P_{3,N3}^2}$(2)}={"CTA","GAC","GAA"}

9.3 H[$\mathbf{P_{3,N3}^2}$(0)] = Ø ，H[$\mathbf{P_{3,N3}^2}$(1)] = Ø ，H[$\mathbf{P_{3,N3}^2}$(2)] = Ø

9.4 由於 H[$\mathbf{P_{3,N3}^2}$(0)]為空集合，故不進行刪除動作

9.5 由於 H[$\mathbf{P_{3,N3}^2}$(0)]為空集合，不進行位置轉換

10. s=s+1=3

11. s=M，執行步驟(6)

12. 使用 Full Search 檢查 P 是否出現在 PP 所列的位置中，檢查後，確定 P 出現在 S 的位置 2。

完整的 Srikantha 方法比對流程如圖 4 所示：

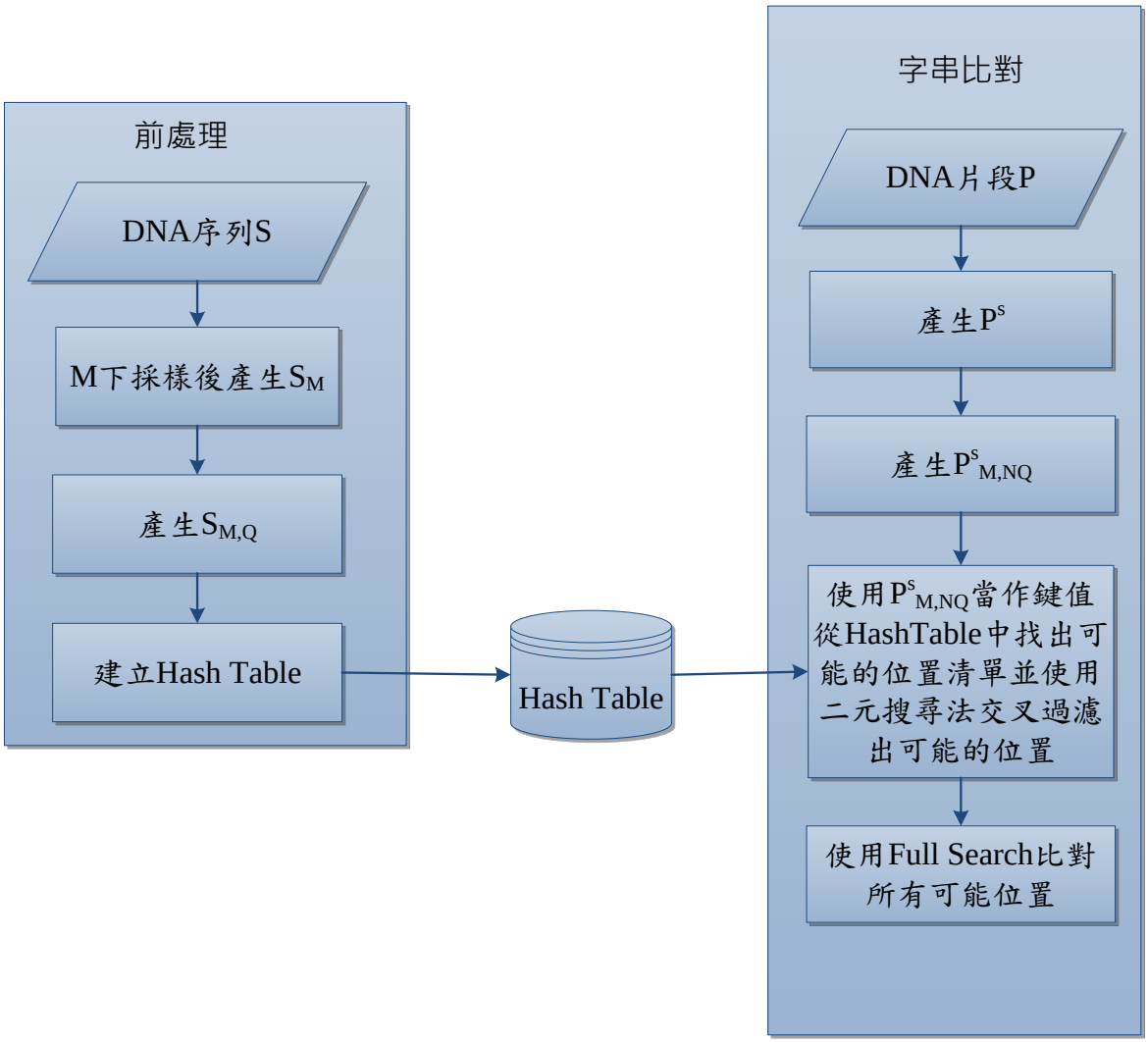

圖 4、Srikantha 方法完整流程圖

第三章、 快速比對方法

本論文提出三個改善方法，分別是針對$L_P > Q \times M$時無法執行之情況所提出的『多連續位置清單擷取方法』，以及為了降低位置清單過濾時間所提出的『線性位置過濾方法』與『去除無效的位置過濾動作』。以下將分別介紹上述三個方法，以及本論文所提的完整 DNA 片段完全比對演算法。

3.1. 多連續位置清單擷取方法

當$L_P > Q \times M$時無法執行的原因是由於片段 P 進行 M 下採樣後，P_M的長度不足建立鍵值所需的長度 Q，因此無法順利產生鍵值至雜湊表(Hash Table)中取出可能的位置清單。

解決的方法是將雜湊表中所有以P_M為開頭的鍵值所對應到的位置清單取出使用。因為雜湊表在建立時，位置清單的儲存方式悉根據鍵值的順序儲存，要取出以 P_M 為開頭的所有可能位置，只需將 P_M 片段長度不足的部分以'A'補足後的片段當作起始鍵值，以'G'補足後的片段當作終止鍵值，便可取出所有以 P_M 為開頭之鍵值所對應到的位置清單。

為了方便理解，以下使用一個簡單的例子說明。

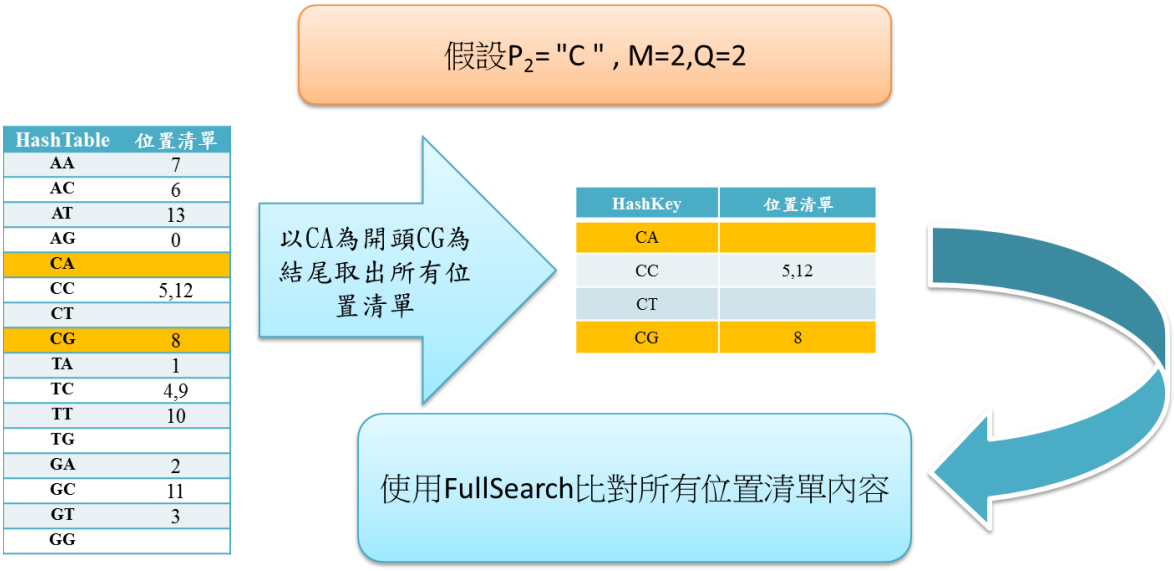

圖 5、多連續位置清單擷取方法範例

由圖 1 所示，有一 P_2= "C "，但其長度不足 Q=2，因此無法產生鍵值，透過使用多位置清單擷取方法，將 P_2 使用'A'補足後產生"CA"當作起始鍵值，使用'G'補足後產生"CG"當作終止鍵值，抓取"CA"到"CG"中的所有位置清單中內容當作可能位置使用 Full Search 進行比較。

3.2. 線性位置過濾方法

在 Srikantha 方法的步驟 3.4 中，使用二元搜尋法來檢查位置清單 $H[\mathbf{P^S_{M,NQ}}(0)]$ 的相對位置是否出現在後續的位置清單中 $(H[\mathbf{P^S_{M,NQ}}(i)]$，$1 \le i < \left\lfloor \frac{\mathbf{L_{P^S_M}}}{\mathbf{Q}} \right\rfloor)$，進而過濾出可能的位置。假設兩個長度分別為 k 及 n 的位置清單，使用二元搜尋法過濾位置的時間複雜度為 $\mathbf{O(k \log n)}$。

由於雜湊表內的所有位置清單都是由小到大排列好的數列，基於這

個特性，我們提出一套線性位置過濾方法來取代二元搜尋法。作法是先從兩個排序好的位置清單中取出第一個位置進行比較，如果兩個位置相同，則將該位置儲存起來，然後再分別由兩個位置清單中取出下一個位置進行比較；如果兩個位置不相同，則從數值較小的位置清單中取出下一個位置進行比較；重覆以上過程，直到任一個位置清單中的位置全部比完為止。令 H[1]與 H[2]是兩個位置清單，$L_{H[1]}$與 $L_{H[2]}$分別是 H[1]與 H[2]內的位置數量，詳細線性位置過濾方法如下：

線性位置過濾演算法

1. 令 PP = Ø
2. 令 i = 0 , j = 0
3. 若 H[1](i)>H[2](j)，則 j=j+1

   若 H[1](i)<H[2](j)，則 i=i+1

   若 H[1](i)=H[2](j)，將 H[1](i)加入 PP，i=i+1，j=j+1
4. 若(i<$L_{H[1]}$)且(j<$L_{H[2]}$)，回到步驟(3)
5. 完成比對，PP 中的位置代表兩序列相同的位置

為了方便了解線性位置過濾演算法，以下使用一個簡單的例子來說明。假設位置清單 H[1]={2,3,10,11}及 H[2]={2,7,10}，使用線性位置過濾

演算法執行步驟如下：

1. PP = Ø
2. i = 0 , j = 0
3. H[1](i)=H[2](j)=2，令 PP = Ø∪{2} = {2} ; i=i+1=1; j=j+1=1
4. H[1](i)<H[2](j)，令 i=i+1=2
5. H[1](i)>H[2](j)，令 j=j+1=2，
6. H[1](i)=H[2](j)=10，令 PP = {2} ∪ {10} ={2, 10}，i=i+1 = 3，j=j+1=3
7. $j=L_{H[2]}$，搜尋完成，PP={2,10}。

使用線性過濾法過濾兩個長度分別為 k 及 n 的位置清單，所需的時間複雜度只需要$\mathbf{O(k+n)}$，相較於使用二元搜尋法進行過濾所需的時間，線性位置過濾法可大幅降低過濾所需的時間。

3.3. 去除無效的位置過濾動作

在 Srikantha 方法中，雜湊表內每個項目的位置清單大小跟$\mathbf{L_S}$、M、與 Q 參數有關，假設每個位置分配至雜湊表內各項目的機率相同，則每個項目的位置清單大小約為$(\frac{\mathbf{L_S}}{\mathbf{M}} \times \frac{\mathbf{1}}{\mathbf{4^Q}})$個位置，經過 i 次過濾後，剩下的位置數約為$(\frac{\mathbf{L_S}}{\mathbf{M}}) \times (\frac{\mathbf{1}}{\mathbf{4^Q}})^{\mathbf{i}}$個。例如：$\mathbf{L_S = 3{\times}10^9}$、M=10、Q=8，則每個項目的位置清單大小約為$(\frac{\mathbf{3{\times}10^9}}{\mathbf{10}} \times \frac{\mathbf{1}}{\mathbf{4^8}})$ ≈4578，經過 1 次過濾後，剩下的位置數不到 1。

事實上，在分析多種合理的$\mathbf{L_S}$、M、Q 參數設定與實際的實驗驗證後發現，經過 1 次過濾後，剩下的位置數通常很少。在位置數不多的情形下進行過濾動作，反而會增加整體計算成本。為了有效改善整體效能，在我們的方法中，只做一次位置過濾動作。只做一次過濾動作，除可避免無實際效益的過濾動作外，還可省下迴圈控制與條件判斷所需的成本。

此外，為了提高過濾效果，在進行過濾動作之前，我們先檢查所有可用的位置清單(H[$\mathbf{P^S_{M,NQ}}$(i)]，0≤i< $\left\lfloor \frac{\mathbf{L_{P^S_M}}}{\mathbf{Q}} \right\rfloor$)，找出兩個位置數量最少的位置清單，然後再使用 3.2 節所介紹的線性位置過濾方法對這兩個位置清單進行過濾。使用 3.2 與本小節所介紹的方法，可有效將 Srikantha 方法的時間複雜度由$\mathbf{O}(\frac{\mathbf{L_S}}{\mathbf{4^Q}}\mathbf{log}\left(\frac{\mathbf{L_S}}{\mathbf{M4^Q}}\right))$降到$\mathbf{O}(\frac{\mathbf{L_S}}{\mathbf{4^Q}})$。

3.4. 快速比對演算法

本論文提出以上三個方法改善 Srikantha 的 DNA 片段比對方法。由於所提方法取出可用位置清單中位置數量最少的位置清單出來使用，因此原 Srikantha 方法的公式(9)與公式(13)必須進行修正。令 H[$\mathbf{P^S_{M,NQ}}(l_1)$]與 H[$\mathbf{P^S_{M,NQ}}(l_2)$]分別是位置最少的兩個位置清單，而且$l_2 > l_1$，則公式(9)與公式(13)分別修正如公式(14)與公式(15)所示：

X+(l_2-l_1)×Q∈H [$\mathbf{P^S_{M,NQ}}(l_2)$] ..... (14)

$$Y = H[\mathbf{P}^{\mathbf{s}}_{\mathbf{M,NQ}}(l_1)](k)-l_1\times Q)\times M-s \quad \text{..... (15)}$$

本論文所提的完整 DNA 快速比對演算法整理如下：

DNA 快速比對演算法

---

1. 令 PP = Ø
2. 令 s=0
3. 若 $L_P<Q\times M$，則使用多位置清單擷取方法，跳至步驟(8)
4. 產生 $\mathbf{P}^{\mathbf{s}}$
   - 4.1 產生 $\mathbf{P}^{\mathbf{s}}$ 下採樣結果 $\mathbf{P}^{\mathbf{s}}_{\mathbf{M}}$
   - 4.2 產生 $\mathbf{P}^{\mathbf{s}}_{\mathbf{M,NQ}}$
   - 4.3 取出 $H[\mathbf{P}^{\mathbf{s}}_{\mathbf{M,NQ}}(l_1)]$與 $H[\mathbf{P}^{\mathbf{s}}_{\mathbf{M,NQ}}(l_2)]$
   - 4.4 使用線性位置過濾法刪除 $H[\mathbf{P}^{\mathbf{s}}_{\mathbf{M,NQ}}(l_1)]$中不符合公式(14)的位置
   - 4.5 使用公式(15)將 $H[\mathbf{P}^{\mathbf{s}}_{\mathbf{M,NQ}}(l_1)]$的位置轉換成相對應於 S 的位置後加入 PP
5. s=s+1
6. 若 s < M，跳到步驟(4)
7. 使用 Full Search 方法檢查 PP 中所有位置
8. 比對完成

為了方便了解序列比對的過程，以下使用一個簡單的例子來說明 DNA 片段搜尋的過程。假設 M=3，Q=3

P="TACATTTGACGGATACACATGTGACCA"

S="CATACATTTGACGGATACACATGTGACCAA"

雜湊表與表一相同。

DNA 片段比對過程如下：

1. PP= Ø
2. s=0
3. L_P> Q×M，故不使用多位置清單擷取方法
4. P^0="TACATTTGACGGATACACATGTGACCA"

   4.1 $\mathbf{P_3^0}$="TATCACATC"

   4.2 $\mathbf{P_{3,N3}^0}$={$\mathbf{P_{3,N3}^0}$(0),$\mathbf{P_{3,N3}^0}$(1) ,$\mathbf{P_{3,N3}^0}$(2)}={"TAT","CAC","ATC"}

   4.3 取出位置數最少的兩個清單 H[$\mathbf{P_{3,N3}^0}$(1)] = Ø ；H[$\mathbf{P_{3,N3}^0}$(2)] = Ø

   4.4 H[$\mathbf{P_{3,N3}^0}$(1)]為空集合，不進行刪除

   4.5 H[$\mathbf{P_{3,N3}^0}$(1)]為空集合，不進行位置轉換
5. s=s+1=1
6. s<M，跳到步驟(4)
7. P^1="ACATTTGACGGATACACATGTGACCA "

   7.1 $\mathbf{P_3^1}$="ATGGTATGC"

7.2 $\mathbf{P^1_{3,N3}}$={$\mathbf{P^1_{3,N3}}$(0),$\mathbf{P^1_{3,N3}}$(1) ,$\mathbf{P^1_{3,N3}}$(2)}={" ATG","GTA","TGC"}

7.3 取出位置個數最少的兩個清單

H[$\mathbf{P^1_{3,N3}}$(1)] = {4}；H[$\mathbf{P^1_{3,N3}}$(2)] = {7}

7.4 位置{4}符合公式(14)，H[$P^1_{3,N3}$(1)]= {4}

7.5 將位置{4}使用公式(15)轉成對於 S 的位置

((4-1×3) ×3–1 = 2)後加入 PP，PP={2}

8. s=s+1=2

9. s<M，跳到步驟(4)

10. P^2="CATTTGACGGATACACATGTGACCA "

10.1 P^2_3="CTAGACGAA"

10.2 $P^2_{3,N3}$={$P^2_{3,N3}$(0),$P^2_{3,N3}$(1) ,$P^2_{3,N3}$(2)}={"CTA","GAC","GAA"}

10.3 取出位置個數最少的兩個清單

H[$P^2_{3,N3}$(0)] = Ø ；H[$P^2_{3,N3}$(1)] = Ø

10.4 由於 H[$P^2_{3,N3}$(0)]為空集合，故不進行刪除

10.5 由於 H[$P^2_{3,N3}$(0)]為空集合，不進行位置轉換

11. s=s+1=3

12. s=M，執行步驟(7)

13. 使用 Full Search 檢查 PP 中所有位置，確定 P 出現在 S 的位置 2。

本論文所提之快速 DNA 完全比對演算法的完整流程如圖 6 所示：

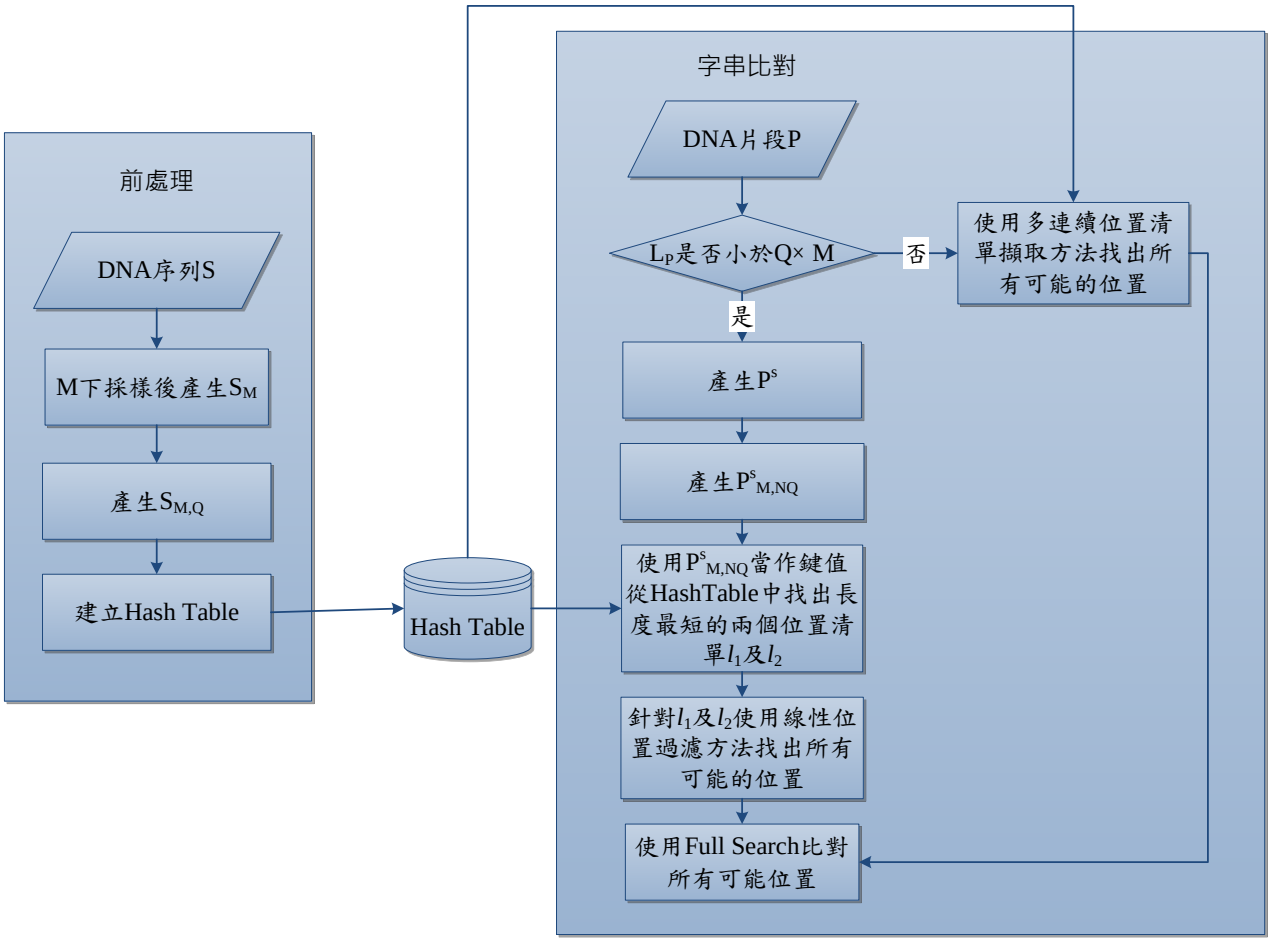

圖 6、本論文所提之演算法完整流程圖

第四章、 實驗結果

4.1.實驗環境

為了比較所提方法與 Srikantha 方法在比對時間上的差異，本實驗使用 UCSC[26]所提供的人體基因序列作為(資料大小為 106M)待比對的 DNA 序列(S)，然後由 DNA 序列中隨機取出所需長度片段各 100 組作為欲比對的 DNA 片段(P)。所有實驗均在同一台個人電腦下執行，電腦配備為 intel Core i5-2400 3.10GHz CPU、2.95G 記憶體、作業系統為 Microsoft Windows 7 Professional 32bit、程式開發環境為 Microsoft Visual Studio C++ 2010。

為了比較在不同 Q-M 組合中對於搜尋速度的影響，實驗設定了 6 種 Q(2,4,6,8,10,11)及 5 種 M(2,4,8,16,32)進行組合測試，藉以了解針對同樣的 S 及 P 在不同的 Q-M 組合中所能改善的效果。

為了驗證本論文所提出的三個方法對於搜尋速度及可用性的影響，將於下列各小節分別單獨使用所提之方法針對 Srikantha 進行改善，並展示實驗結果。

4.2.多連續位置清單擷取方法執行效能分析

影響多連續位置清單擷取方法在比對速度上最大的因素取決於P_M的長度，為了解不同P_M長度下，多連續位置清單擷取方法的效能表現，本實驗設定 Q=6，M=16，5 種L_P(16、32、48、64、80)，可以產生P_M長度(L_{P_M})為 1~6 的情況，並將其所需比對時間與第二章所提到的快速字串比對演算法及 Full Search 進行比較。表 3 為使用本方法從序列 S 中針對每種L_P的 100 組片段 P 各進行 10 次比對後的平均時間，時間單位為毫秒(ms)。表 4 為第二章所提到的演算法及 Full Search 針對每種L_P的 100 組片段 P 各進行 10 次比對後的平均時間，時間單位為毫秒(ms)。

表 3、多連續位置清單擷取方法比對所需時間(ms)

L_P(L_{P_M})	搜尋時間
L_P=16 (L_{P_M}=1)	1161.820
L_P=32 (L_{P_M}=2)	293.617
L_P=48 (L_{P_M}=3)	73.765
L_P=64 (L_{P_M}=4)	18.518
L_P=80 (L_{P_M}=5)	4.84

表 4、字串比對演算法比對所需時間(ms)

	L_P=16	L_P=32	L_P=48	L_P=64	L_P=80
Full Search	631.585	622.416	631.641	636.106	632.845
Knuth Morris Pratt (KMP)	646.823	635.162	644.329	651.711	647.353
Karp Rabin (KR)	314.693	314.969	314.882	314.927	314.967

由表 3 及表 4 的結果顯示，只有在多位置清單擷取方法在L_{P_M}=1 的時候，其搜尋結果較 Full Search 差，因此在未來的設計上，在判斷L_{P_M}=1 時，可以直接使用 Full Search 或是其他快速的字串比對演算法。當 $Q>L_{P_M}>1$ 時，多連續位置清單擷取方法可以使原本 Srikantha 演算法無法運作的情況得以解決，且在大多數情況下所花費的時間較 Full Search 來的低，因此證明此方法可以有效的提高程式的可用性。

由表 3 的結果顯示，KMP 演算法在執行速度上，較 Full Search 來的緩慢，是由於其所建立起的失誤函數內容大多為-1，因此在比對失敗的時候，無法從失誤函數中取得較多的好處，且還需要計算位移量，因此 KMP 演算法在此的效果相較於 Full Search 差。在 KR 的執行速度上，其效率會隨著雜湊函式的設計方式而有所不同，在本論文雜湊函式的設計上，可以使得 KR 的效率較 Full Search 來的快速，但與多連續位置清單擷取方法相比較，KR 演算法只有在L_{P_M}=1 時可以得到較好的效果。

4.3.線性位置過濾方法執行效能分析

為了驗證使用線性位置過濾方法能夠有效降低過濾所需的時間，本實驗分別利用 Srikantha 方法與本論文所提的線性位置過濾方法進行DNA比對。表 5 為使用 Srikantha 的方法，在不同 Q-M 組合下，從序列 S 中針對 100 組長度 300 的不同片段 P 各進行 10 次比對後的平均時間，時間單位為毫秒(ms)。表 6 為使用線性位置過濾改良過後的方法，在不同 Q-M 組合下，從序列 S 中針對 100 組不同的片段 P 各進行 10 次比對後的平均時間。表 7 為 Srikantha 方法使用線性位置過濾方法改良後所需時間相較於 Srikantha 演算法可以降低比對所需時間的百分比(計算公式：$\frac{\text{改良前所需時間}-\text{改良後所需時間}}{\text{改良前所需時間}}\times 100\%$)。

表 5、Srikantha 演算法比對所需時間(ms)

	M=2	**M=4**	**M=8**	**M=16**	**M=32**
Q=2	576.985	552.732	533.724	520.143	501.932
Q=4	31.338	28.644	27.284	26.822	26.622
Q=6	1.877	1.715	1.656	1.58	0.884
Q=8	0.114	0.108	0.104	0.099	0.079
Q=10	0.013	0.017	0.015	0.016	---
Q=11	0.01	0.011	0.011	0.013	---

表 6、使用線性位置過濾方法改良後比對所需時間(ms)

	M=2	**M=4**	**M=8**	**M=16**	**M=32**
Q=2	341.413	189.111	151.431	141.814	121.687
Q=4	13.913	8.226	6.641	6.285	5.364
Q=6	0.699	0.447	0.395	0.393	0.893
Q=8	0.047	0.037	0.035	0.034	0.081
Q=10	0.01	0.012	0.013	0.018	--
Q=11	0.01	0.012	0.012	0.011	--

表 7、線性位置過濾方法可以降低所需時間之百分比

	M=2	**M=4**	**M=8**	**M=16**	**M=32**
Q=2	41%	66%	72%	73%	76%
Q=4	56%	71%	76%	77%	80%
Q=6	63%	74%	76%	75%	-1%
Q=8	59%	66%	66%	66%	-3%
Q=10	23%	29%	13%	-13%	---
Q=11	0%	-9%	-9%	15%	---

由表 7 的結果顯示，可以看到大部分情況下線性位置過濾方法都可以減少 50%以上的執行時間。當在 Q×M 越趨近於 L_P 時，效果越差，其主要原因是位置清單中的位置越少，線性位置過濾方法(時間複雜度 O(k+n))與二元搜尋法(時間複雜度 O(k log n))的比對時間越接近。由於

Q×M 越大，每個位置清單中的位置越少，因此會造成效果較差的情況產生。使用線性位置過濾方法結合 Srikantha 方法後，在大多數的情況下都可以降低比對所需要的時間，因此證明此方法是可以有效降低 Srikantha 的比對時間。

4.4.去除無效的位置過濾動作執行效能分析

為了驗證 Srikantha 搭配使用本論文所提出的去除無效的位置過濾動作可以有效降低過濾所需的時間，本論文透過實驗 Srikantha 比對所需時間與經過去除無效的位置過濾動作改良後 Srikantha 演算法所需之比對時間的差異。表 8 為 Srikantha 使用去除無效的位置過濾動作改良後的方法，在不同 Q-M 組合下，從序列 S 中針對 100 組長度為 300 的不同片段 P 各進行 10 次比對後的平均時間，時間單位為毫秒(ms)。表 9 為 Srikantha 方法使用去除無效的位置過濾動作改良後所需時間相較於 Srikantha 演算法可以降低比對所需時間的百分比。

表 8、去除無效的位置過濾動作改良後比對所需時間(ms)

	M=2	**M=4**	**M=8**	**M=16**	**M=32**
Q=2	358.432	354.106	342.132	355.374	376.778
Q=4	15.273	16.291	16.859	18.851	22.636
Q=6	0.811	0.886	0.942	1.088	0.876
Q=8	0.049	0.056	0.065	0.089	0.084
Q=10	0.01	0.012	0.016	0.018	---
Q=11	0.011	0.01	0.011	0.014	---

表 9、去除無效的位置過濾動作可以降低所需時間之百分比

	M=2	**M=4**	**M=8**	**M=16**	**M=32**
Q=2	38%	36%	36%	32%	25%
Q=4	51%	43%	38%	30%	15%
Q=6	57%	48%	43%	31%	1%
Q=8	57%	48%	38%	10%	-6%
Q=10	23%	29%	-7%	-13%	---
Q=11	-10%	9%	0%	-8%	---

由表 8 的結果顯示，在 L_P 及 Q 相同的設定下，其比對所需時間隨著 M 的遞增而遞增的主要因素，是由於當 M 越大時，需要執行 M 次判斷P_M^S產生出的 NQ-Gram 個數是否大於 2，且須針對每個P_M^S找尋長度最短的兩個位置清單，但每個P_M^S所能刪除的 NQ-Gram 卻相較於 M 小時少。因此當 M 越大需要執行判斷的次數越多，且帶來的效益越小，因此比對時間才會隨著 M 的遞增而遞增。

由表 9 的結果顯示，當 Q×M 越大的時候，效果越差，其主要原因是在於當 Q×M 越大時每個P_M^S中的 NQ-Gram 個數將會越趨近於 1，因此去除無效的位置過濾動作不但無法刪除任何 NQ-Gram，且還需要負擔判斷及找尋最短位置清單的時間。但使用去除無效的位置過濾動作在大多數

的情況下仍可以降低比對所需要的時間。

4.5.快速基因比對演算法分析

為了驗證本論文所提出的所有改善方法結合後，可以有效的降低比對所需之時間及提高可用性，表 10 為使用本論文所提出的方法在不同 Q-M 組合下,從序列 S 中針對 100 組長度為 300 的不同片段 P 各進行 10 次比對後的平均時間，時間單位為毫秒(ms)。表 11 為本論文所提出的演算法所需時間相較於 Srikantha 方法可以降低比對所需時間的百分比。

表 10、所提演算法之比對所需時間(ms)

	M=2	**M=4**	**M=8**	**M=16**	**M=32**
Q=2	32.624	32.535	33.794	36.909	42.184
Q=4	1.556	1.698	1.800	2.070	2.587
Q=6	0.092	0.11	0.121	0.147	0.434
Q=8	0.010	0.011	0.013	0.021	0.044
Q=10	0.007	0.007	0.007	0.008	0.016
Q=11	0.005	0.006	0.006	0.008	0.015

表 11、所提方法可以降低所需時間之百分比

	M=2	**M=4**	**M=8**	**M=16**	**M=32**
Q=2	94%	94%	94%	93%	92%
Q=4	95%	94%	93%	92%	90%
Q=6	95%	94%	93%	91%	51%
Q=8	91%	90%	88%	79%	44%
Q=10	46%	59%	53%	50%	---
Q=11	50%	45%	45%	38%	---

由表 11 的結果顯示，本論文所提的方法在所有 Q-M 組合的情況下，皆可有效減少 Srikantha 比對的速度，在最好的情況下可以減少約 95%的比對時間，最差情況下也可以減少約 38%的比對時間。

由表 5 及表 10 比對後顯示，在 Q 的設計上，本論文所提的方法可以使用較小的 Q 來達到與 Srikantha 相同的比對時間。當 Q 越小時，雜湊表建立時所需要的空間(4^Q)越少，因此以相同的比對時間來看，所提的方法相較於 Srikantha 可以使用較少的空間來完成。且本論文所提的方法在 $Q \times M > L_P$(M=32，Q=10、11)的情況下，程式也可以正常運作，改善了 Srikantha 無法執行之狀況。

綜合比較表 7、表 9、與表 11，我們可以發現整合『線性位置過濾方法』與『去除無效位置過濾動作』兩個方法可有效降低整體執行時間，且在所有 Q-M 組合下都能得到很好的效果。

第五章、 結論

隨著人體基因定序成本日益降低，如何有效使用基因比對找出需要的片段變得越來越重要。完全比對應用於基因序列，可以用來找尋特定的基因片段，或是在短縱列重複序列(STR)的相關應用上，如找尋是否有過度擴增的 STR 序列來檢測是否有相關性疾病等。

Srikantha 是目前完全比對應用於 DNA 序列中最快速的方法，Srikantha 方法使用雜湊表加速找尋片段 P 存在於序列 S 中位置的速度，並使用 M 下採樣減少序列 S 的長度，以及過濾掉位置清單中不可能的位置，降低所需的使用空間及加速了比對的時間。

本論文使用『多位置清單擷取方法』提高演算法的可用性，另外使用了『線性位置過濾方法』與『去除無效的位置過濾動作』兩個方法改善 Srikantha 的比對速度，使得原先需要大量時間的過濾過程變的快速，不論在任何 Q-M 的組合下，皆可表現出較佳的效果。結合以上所提之方法，本論文提出了一套新的 DNA 完全比對演算法，可以有效的改善現有最快速之完全比對演算法約 38%~95%的比對時間。

當應用於現實的情況中，一個完整的人體序列 S 長度有可能高達 30 億個鹼基，並且需要與多個片段 P 進行比對，其中部分片段 P 的長度有可能高達數萬個鹼基。當 L_P 越長時，其與 Q×M 的差異就有越大，本論文在此情況下相對於 Srikantha 演算法越能表現出較好的效果。因此在總

體的比對時間上，可以大幅度的降低基因檢測所需的時間。

參考文獻

[1] 鍾竺均與陳偉(2007)，生物技術概論，新文京開發出版社。

[2] Robert A. Holt and Steven J.M. Jones (2008), "The new paradigm of flow cell sequencing," Genome Research, 18(6), pp.839 -46.

[3] Human Genome Project Information, http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml.

[4] National Human Genome Research Institute, National Institute of Health, http://www.genome.gov.

[5] Archon Genomics X PRIZE, http://genomics.xprize.org.

[6] Roche, http://www.roche.com.tw.

[7] Illumina, http://www.illumina.com.

[8] Life Technologies, http://www.lifetechnologies.com.

[9] Pacific Biosciences, http://www.pacificbiosciences.com.

[10] Jonathan M. Rothberg et al. (2011), "An integrated semiconductor device enabling non-optical genome sequencing," Nature, 475(7356), pp.348-352

[11] Holly A. Hammond, Li Jin, Y. Zhong, C. Thomas Caskey and Ranajit Chakraborty (1994), "Evaluation of 13 short tandem repeat loci for use in personal identification applications," The American Journal of Human Genetics, 55(1), pp.175-189.

[12] Cheng Hwai Tzeng et al. (1998), "Polymorphisms of twelve short tandem repeat loci in a Taiwanese population and their application in parentage testing," Journal Of The Formosan Medical Association, 97(11), pp.738-744.

[13] Susan E. Andrew et al. (1993), "The relationship between trinucleotide repeat length (CAG) and clinical features of Huntington disease," Nature Genetics, 4(4), pp.398-401.

[14] S. Rajesh, S. Prathima and Dr.L.S.S. Reddy (2010), "Unusual Pattern Detection in DNA Database Using KMP Algorithm," International Journal of Computer Applications, 1(22), pp.1-7.

[15] Abhilash Srikantha, Ajit S. Bopardikar, Kalyan Kumar Kaipa, Parthasarathy Venkataraman, Kyusang Lee, TaeJin Ahn and Rangavittal Narayanan (2010), "A fast algorithm for exact sequence search in biological sequences using polyphase decomposition," Bioinformatics, 26(18), pp.414-419.

[16] V. I. Levenshtein (1966), "Binary codes capable of correcting deletions, insertions, and reversals," Soviet Physics-Doklady, 10(8), pp.707-710.

[17] Xiaoqiu Huang and Kun-Mao Chao (2003), "A generalized global alignment algorithm," Bioinformatics, 19(2), pp.228-233.

[18] Li Yujian and Liu Bo (2007),"A Normalized Levenshtein Distance Metric," IEEE Transactions on Pattern Analysis and Machine Intelligence,29(6), pp.1091-1095.

[19] Robert S. Boyer and J. Strother Moore (1977), "A fast string searching algorithm," Communications of the ACM, 20(10), pp.762-772.

[20] Donald E. Knuth, James H. Morris, JR. and Vaughan R. Pratt (1977), "Fast pattern matching in strings," SIAM Journal on Computing, 6(2), 323-350.

[21] R. Nigel Horspool (1980), "Practical Fast Searching in Strings," Software – Practice and Experience, Software-Practice and Experience, 10(6), pp.501-506.

[22] Richard M. Karp and Michael O. Rabin. (1987), "Efficient randomized pattern-matching algorithms," IBM Journal of Research and Development, 31(2), pp. 249-260.

[23] Ricardo A. Baeza-Yates and Gaston H. Gonnet (1992), "A new approach to text searching," Communication of the ACM, 35(10), pp.74-82.

[24] Martin Farach (1997), "Optimal Suffix Tree Construction with Large Alphabets," 38th IEEE Symposium on Foundations of Computer Science (FOCS '97), pp.137-143.

[25] Zemin Ning, Anthony J. Cox and James C. Mullikin (2001), " SSAHA: A Fast Search Method for Large," Genome research, 11(10), pp.1725-1729.

[26] UCSC, http://genome.ucsc.edu.