一個簡單的基因群組搜尋演算法
A Simple Algorithm for Finding Gene Clusters

呂威甫*
亞洲大學
資訊工程學系
生物資訊系
weifu@asia.edu.tw

陳彥嘉
亞洲大學
生物資訊系
m0955293618@hotmail.com

* 為通訊作者

摘要

物種演化中大規模的演化事件，基因重組，改變了染色體上基因的順序。通常兩個親源接近的真核生物，有許多基因內容相同但順序的不同的染色體片段，稱為基因群組。基因群組可能是共同祖先演化之後的結果，也可能表現共同調控的現象。其對於演化樹的建構與基因功能的預測都扮演了重要的角色。因此基因群組的尋找，在比較基因體學的研究中，是一個重要的問題。本論文將以近似共同區間討論基因群組的尋找的問題，也就是，基因體將以字串表示，同時近似共同區間將以集合間的對稱差定義。我們提出了一個 $O(n^3)$ 的演算法解決尋找群組的問題。

關鍵詞：比較基因體學、基因重組、基因群組、共同區間

Abstract

Genomes evolve through large-scale events, known as genome rearrangements, that reorganize the gene order in the chromosome. Usually two closely related prokaryotes share many gene clusters, which are sets of genes in close proximity to each other, but not necessarily contiguous nor in the same order in both genomes. Gene clusters could result from evolution of common ancestors, and could represent phenomenon of co-expression. It plays an important role in the construction of phylogenetic trees and prediction of gene functions. Thus, finding gene clusters is an important problem in comparative genomics. In this paper, we will discuss gene clusters finding problem using model of approximate common intervals, that is, genomes are considered as strings and approximate common intervals are defined as set symmetric difference. We present an $O(n^3)$ algorithm to find gene clusters

Keywords: comparative genomics, genome rearrangement, gene cluster, common interval

1. 前言

在生物演化的過程中，物種會因為基因體產生變化的關係演化成其它的物種。所產生的變化包含規模較小的改變，像在 DNA 序列中的點突變，與規模較大的改變，例如大段染色體的重新排列的基因體重組。

基因重組改變了基因的順序，但基因順序的改變並不是隨機的。通常兩個親源接近的原核生物，會有許多基因內容相同但順序的不同的染色體片段，稱為保留基因群組(conserved gene cluster)或稱為基因群組(gene cluster)。所謂保留的意義，指的是這些基因群組保留了生命的內容，可能是共同祖先演化之後的結果[9]。此外，基因群組也可能表現共同調控的現象[10]，由此可以幫助基因預測、蛋白質預測或者合理的去確認基因功能的群組[13]。基因群組的尋找對於生物與資訊而言都是一個具有挑戰性的問題，這個問題在比較生物學、基因註解與演化樹建構上都有很重要的應用

[3]，[4]，[5]，[6]，[7]。

在基因群組的研究中，不同的定義會有不同的結果。第一個對基因群組做明確定義的是 Uno 與 Yagiura [12]，他們以共同區間的概念定義基因群組，考慮將基因體以排列表示，而共同區間則定義為兩個排列間包含相同元素的連續區間。Uno 與 Yagiura [12]提出了在兩個 n 個元素的排列間找出 K 個共同區間的 $O(n+k)$ 時間演算法。

然而，以排列表示基因體並不符合生物實際的情形，因為在基因體上基因可能會重複出現。Schmidt 與 Stoy [11] 考慮將基因體以字串表示，允許表示旁系同源基因(paralogous gene)的字元可以重複出現。他們提出了 $O(n^2)$ 時間的演算法找出兩個長度為 n 的字串間所有的共同區間。

此外，基因群組之間也可能插入其他非基因群組的基因。因此，共同區間的定義必須加以推廣，進一步定義兩個字串間的近似共同區間。也就是，在字串中的連續區間包含了相近元素的則視為近似共同區間。

本篇論文，將以近似共同區間問題討論基因群組的群找的問題。我們考慮以字串表示基因體，並以集合間對稱差定義近似共同區間。我們提出了一個 $O(n^3)$ 的演算法，這個方法利用命名法則的觀念[1]，將比 Amir 等人[2]所提出的方法更為簡單。

本文後續的章節將組織如下。下一節介紹本論文所用到的基本定義，並以較嚴謹的方式描述我們所討論的基因群組與近似共同區間問題。在第三節中，我們討論當 $K = 1$ 的情形下的近似共同區間演算法，並分析其時間複雜度。第四節我們將 $K = 1$ 的演算法推廣至 $K > 1$ 的情形，這是對所有 K 值都成立的較一般化的情形。最後一節是結論，將對我們的論文做一個總結，並探討未來可以進行的研究方向。

2. 基本定義與問題描述

給定一條定義在有限字母集Σ上的字串S。$|S|$為字串S的長度，$S[i]$表示S的第i個字元，且$S[i, j]$表示S中第i個字元開始到第j個字元結束的子字串。在我們的論文中所討論的字串對應於我們所要比對的基因體，而字元則對應於基因體上的基因。對於子字串$S[i, j]$，考慮在這個子字串上面所出現的字的集合，稱為字元集合，定義如下。

定義2.1（字元集合）
給定一個字串S，一個子字串$S[i,j]$的字元集合定義為

$$CS(S[i, j]) \coloneqq \{S[k] \mid i \le k \le j\} \subset \Sigma \text{。}$$

字元集合表示了出現在染色體上給定區間的所有基因，而基因出現的前後次序與次數將被忽略而不考慮。本文中子字串$S[i,j]$將以區間$[i,j]$表示。字串S中不同的區間可能會對應相同的字元集合C，這些區間稱為字元集合C所出現的位置。

定義2.2（最大字元集合）
給定一個字串S，令字元集合C出現在區間$[i,j]$的字元集合。則C稱為最大字元集合，若且唯若 $S[i-1] \notin C$ 且 $S[j+1] \notin C$ 。

一般而言，考慮以字串間的共同區間定義基因群組。

定義2.3（共同區間）
給定兩條字串 S 與 S'。若 $S[i, j]$ 與 $S'[i', j']$ 兩個區間的字元集合相同，則稱字串 S 上的區間 $[i, j]$ 與字串 S' 上的區間 $[i', j']$ 為共同區間。

若考慮在基因群組的基因之間也可能插入其他非基因群組的基因，則需考慮近似共同區間。我們以集合對稱差Δ[1]，定義近似共同區

[1]若 A 與 B 為集合，則 A 與 B 的對稱差，$A \Delta B$，定義為 $\{x \mid (x \in A \text{且} x \notin B) \text{或} (x \in B \text{且} x \notin A)\}$。

間。

定義2.4（近似共同區間）
令CS為出現在字串S上的字元集合，且CS'為出現在字串S'上的字元集合。若兩個集合的對稱差$CS\Delta CS' \le K$，稱CS與CS'為相差K的近似共同區間。

本文將考慮以相差 K 的共同區間定義基因群組，因此我們所要處理的問題定義如下。

定義2.5（近似共同區間問題）
輸入:L 條字串的集合 S=$\{S_1, \ldots, S_L\}$, 與常數 K。
輸出:在S上所有的字元集合，與這些字元集合中所有對稱差為K的關係。

我們將上述概念，以如下的例子加以解釋。令 S = { S_1 = abacb , S_2 = bacab}。S_1所形成的字元集合{a}出現在[1,1], [3,3]; {b}出現在[2,2], [5,5]; {a, b}在[1, 3]; {a, c}在[3, 4] ; {b,c}在[4, 5]和{a, b, c}在[1, 5]. S_2所形成的字元集{a}在[2,2] , [4, 4]; {b}在[1, 1], [5, 5];{c}在[3, 3]; {a, b}在[1, 2] , [4, 5]; {a, c}在[2, 4]和{a, b, c}在[1, 5]。考慮 K=1 的情形，則上述字元集合的近似共同區間之關係為：

{a} ↔ {a, b}, {a, c}
{b} ↔ {a, b}, {b, c}
{c} ↔ {a, c}, {b,c}
{a, b} ↔ {a}, {b}, {a, b, c}
{a, c} ↔ {a}, {c}, {a, b, c}
{b, c} ↔ {b}, {c}, {a, b, c}
{a, b, c} ↔ {a, b}, {b, c}, {a, c}

本文我們提出一個簡單的$O(n^3)$ 演算法，解決近似共同區間問題。下一節我們將討論處理K = 1的特殊情形，之後再將這個結果推廣至 k >1處理的情形。

3. K=1 的近似共同區演算法

我們的演算法將所擷取出的字元集合以 01 字串表示，我們稱這個 01 字串為字元集合的編碼字串。假設$\sum = \{a_1, a_2, ..., a_{|\Sigma|}\}$，則字元集合 CS 將表示成 01 字串$b_1 b_2 ... b_{|\Sigma|}$，其中若$a_i \in CS$則$b_i = 1$，否則$b_i = 0$。例如，$\Sigma = \{a, b, c, d, e, f, g, h\}$，則字元集合{$b$, c, e, g, h}將編碼成01101011。要計算兩個字元集合的對稱差，也就是要求兩個字元集合所對應的 01 字串的漢明距離(Hamming distance)，底下皆稱為 HD。所謂漢明距離指的是給定兩個長度相同的 01 字串，若其中有 k 個位置的 01 不同，則其 HD = k。例如，HD(01100111,01101110) = 2。因此，要找出字串間所有對稱差為 1 的關係，就是要找出所有 01 編碼字串間 HD = 1 的關係。

要求出 01 字串集合間 HD = 1 的關係，最直接的方式就是將集合中所有的字串對計算其 HD，而計算 HD 的方式則是一位元一位元加以比較。若要比較的字串集合共有 m 條長度為 n 的 01 字串，則共有 $O(m^2)$對 01 字串需要比較，且每一對的比較需要花 $O(n)$的時間，共花 $O(m^2 n)$的時間。在最差的情況下 m 約等於 $O(n^2)$ [2]，所以總共花 $O(n^5)$。這是很沒有效率的。

本篇我們使用有效率的方式決定字元集合間 HD = 1 的關係，主要根據如下的簡單的概念。

定義 3.1 ($L(X)$ 和 $R(X)$)
令 $X = x_1 x_2 ... x_n$ 為 01 字串，則 $X[1,n/2]$稱為 X 的左半部，以符號 $L(X)$表示；$X[n/2+1,n]$稱為 X 的右半部，以符號 $R(X)$表示。若 X 為 1 個字元的字串，則 $L(X)$＝$R(X)$ = X。
例:令 X = 11011100，則 $L(X)$＝1101，$R(X)$ = 1100。

引理 3.1
令 $X = x_1 x_2 ... x_n$ 與 $X = x_1 x_2 ... x_n$ 為 01 字串。
$HD(X,Y)$=1 若且唯若
(1) $HD(L(X), L(Y)) = 0$ 且 $HD(R(X), R(Y)) = 1$.
或 (2) $HD(L(X),L(Y)) = 1$ 且 $HD(R(X), R(Y)) = 0$.

證明:
根據漢明距離定義

$HD(X,Y) = HD(L(X), L(Y))+HD(R(X), R(Y))$

當 $HD(X,Y)= 1$，也就是 $HD(L(X),L(Y)) + HD(R(X), R(Y))= 1$。因為漢明距離大於等於 0，所以滿足 $HD(L(X),L(Y)) + HD(R(X), R(Y)) =1$ 只有兩種情形：

$HD(L(X), L(Y)) =0$ 且 $HD(R(X), R(Y)) =1$ 或

$HD(L(X), L(Y)) =1$ 且 $HD(R(X), R(Y)) =0$. ■

根據以上的引理，要判別兩個 01 字串的 *HD* 是否為 1，只需分別比較這兩個字串左半 *L*(*X*)與 *L*(*Y*) *HD* 以及右半 *R*(*X*)與 *R*(*Y*)的 *HD*。若左半差 1 右半相同或左半相同右半差 1，這兩個字串的 *HD* 才會為 1。至於如何知道左半與右半是否差 1，則可以將字串的左半與右半在分別拆成左半與右半遞迴比較下去，直到底層。但是遞迴的比較將會非常浪費時間，本篇我們使用疊代的方式決定字串集合間 *HD*=1 的關係。以下將使用一個命名法則來命名 01 字串，使得每個 01 字串與其左半與右半都有唯一的命名編號，以方便我們做 *HD*=1 的比對。

3.1 命名法則

命名的法則的概念是把兩個數字一組，給定一個常數編號。若這一組數字之前曾經出現，則將此組數字編碼成之前所定義的編號。若遇到的是新的一組數字，則將目前的編號+1 給其新的編號。編號號碼從 2 開始編號避免跟元素出現與否的 01 字串搞混。若字母集Σ的大小不是 2^k，則我們可以在不滿 2^k 的部份補 0 方便命名。

考慮長度為 *n* 的 01 字串，命名法則執行方法是在最底層放上 01 字串，把這個字串兩個字元一組給定一個編號，依序將這個 01 字串編碼完，分別得到 *n*/2 個編碼。接著將這 *n*/2 個編碼，兩個一組給定一個編號，得到 *n*/4 個編碼。這樣依序下去編碼到最上層，直到可以以一個編號來代表 0,1 字串。

例如，考慮如表 1 的字串 0110011 的命名方式，在底層放上 01 字串，在兩個位元組成層 0,1 給一個編號 2，當出現 1,0 給一個編號 3，在四個位元組成層 2,3 給一個編號 5。依照這個原則到可以用一個編號來表示這個子字串，也就是四位元組成層 5,6 給一個編號 7。

表一 命名編範例

<table>
<tr><td colspan="8">7</td></tr>
<tr><td colspan="4">5</td><td colspan="4">6</td></tr>
<tr><td colspan="2">2</td><td colspan="2">3</td><td colspan="2">3</td><td colspan="2">4</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
</table>

在這個命名編號中使用二元樹結構來記錄編號是否出現，另外還使用一個陣列來記錄(見表 2)，*X* 欄存放 01 字串，*Id*(*X*)存放代表 x 字串的編號，*L*(*X*)儲存組成 *X* 的左半編碼，*R*(*X*)儲存組成 *X* 的右半編碼，*Len*(*X*)為 01 字串的長度。

表二 命名編號組成表

X	*Id*(*X*)	*L*(*X*)	*R*(*X*)	*Len*(*X*)
01	2	0	1	2
10	3	1	0	2
11	4	1	1	2
0110	5	2	3	4
1011	6	3	4	4
01101011	7	5	6	8
1111	8	4	4	4
01101111	9	5	8	8

命名編號組成表中的每一個字串都有唯一對應的編號，因此，本文中討論將字串與編號將視為相同。我們使用[8]的方法找出字串集合上所有的字元集合，並加以編碼。接下來我們使用這個命名編碼，決定 01 字串集合間 *HD* ＝1 的關係。

3.2 演算法

當我們得到命名編號組成表之後，我們使用疊代的方式決定字串集合間 HD=1 的關係。在演算法之前，我們需要一些相關的定義。

定義 3.2 (分割集合 $Sp(S)$)
令 S 為一個 01 字串所形成的集合，則 S 的分割集合 $Sp(S)$，定義為 $\{L(X), R(X) \mid X \in S\}$。

例：令 S ={00110000, 10101100, 11111001}，則 $Sp(S)$ = {0011, 0000, 1010, 1100, 1111, 1001}

一個 01 字串集合 S 的分割集合 $Sp(S)$，是取集合 S 中每一個字串的左半部 $L(X)$與右半部 $R(X)$部所形成的集合。因此 $Sp(S)$仍是一個 01 字串的集合，我們可以再對 $Sp(S)$取分割集合，得到新的集合 $Sp(Sp(S))$，此集合以 $Sp^{(2)}(S)$表示。同理 $Sp^{(2)}(S)$仍可繼續取分割集合。

定義 3.3 $Sp^{(*)}(S)$
定義 $Sp^{(n)}(S) = Sp(Sp^{(n-1)}(S))$，若 $n>1$；$Sp^{(n)}(S) = Sp(S)$，若 $n=1$。同時，$Sp^{(*)}(S) = \bigcup_{i=1}^{\infty} Sp^{(i)}(S)$。

例：令 S ={00110000, 10101100, 11111001},則 $Sp^{(2)}(S)$ = {00,11,10,01}，$Sp^{(3)}(S)$={0,1}，$Sp^{(*)}(S)$ = {0,1,00,11,10,01, 0011, 0000, 1010, 1100, 1111, 1001}

令 S_{cs} 為所有出現的字元集合的集合，則 $Sp^{(*)}(S_{cs})$為命名編號表中所有分割字串集合的聯集。

定義 3.4 (ℓ 位元集合)
考慮由命名編號組成表中所有長度為 ℓ 的字串所形成的集合稱為 ℓ 位元集合 SB_ℓ，也就是 $SB_\ell = \{X \in Sp^{(*)}(S_{cs}) \mid Len(X) = \ell\}$。

例: SB_2 = {2, 3, 4}，SB_4={5, 6, 8}。

定義 3.5 (ℓ 位元關係圖)
考慮 ℓ 位元集合 SB_ℓ 間漢明距離為 1 的關係，將此關係定義為 ℓ 位元關係圖 $G_\ell = (V_\ell, E_\ell)$。其中 $V_\ell = SB_\ell$，V_ℓ 中的任兩個點 v_i, v_j 之間有邊若且唯若 $HD(v_i, v_j) = 1$。同時，給定圖上的每個邊 $e = (v_i, v_j)$一個值 $w(e) = c$，表示 v_i 與 v_j 在第 c 個字元不同。

演算法主要的概念是，若我們可以知道命名編號組成表中 SB_ℓ 所有 HD＝1 的關係，我們就可以利用這個關係以及引理 3.1 的性質，決定 $SB_{2\ell}$ 所有 HD＝1 的關係。也就是我們可以由 ℓ 位元關係圖建構 2ℓ 位元關係圖。我們由 1 位元關係圖得到 2 位元關係圖，再由 2 位元關係圖得到 4 位元關係圖，依序疊代下去，最後由 n/2 位元關係圖得到 n 位元關係圖。

我們在這裡描述當我們有 ℓ 位元關係圖之後，如何利用其建構 2ℓ 位元關係圖。在敘述演算法之前，先定義一個函數跟兩個資料結構。

在命名編號表中，X 是由 $L(X)$跟 $R(X)$所組成，我們稱 $Id(L(X))$為編號 $Id(X)$的左編號，$Id(L(X))$為 $Id(X)$的右編號，同時令 $Id(L(X),R(X))$為 $Id(X)$。我們蒐集命名編號表中所有左編號同為 i 的編號的右編號，定義為集合 $Id(i,*)$。也就是，

$Id(i,*)$ = $\{j \mid Y \in Sp^{(*)}(Scs)$, 其中 $L(Y) = i$ 且 $R(Y) = j\}$。

同理，我們也蒐集命名編號表中所有右編號同為 j 的編號的左編號，定義為集合 $Id(*, j)$。也就是，

$Id(*, j)$ = $\{i \mid Y \in Sp^{(*)}(Scs)$, 其中 $L(Y) = i$ 且 $R(Y) = j\}$。

對於 $Id(i,*)$（相對於 $Id(*, j)$）演算法分別用兩個資料結構 $Array(Id(i,*))$ 與 $List(Id(i,*))$（相對於 $Array(Id(*, j))$ 與 $List(Id(*, j))$）來存放，目的是用來加速比對的速度。其中 $Array(Id(i,*))$ 是一個陣列定義為 $Array(Id(i,*))[j] = 1$ 若 $j \in Id(i,*)$，否則 $Array(Id(i,*))[j] = 0$。而 $List(Id(i,*))$ 是一個串列儲存所有集合 $Id(i,*)$ 的元素。

由 $G_\ell = (V_\ell, E_\ell)$ 建構 $G_{2\ell} = (V_{2\ell}, E_{2\ell})$ 的演算法

定義如下。

演算法 3.1 由 $G_\ell = (V_\ell, E_\ell)$ 建構 $G_{2\ell} = (V_{2\ell}, E_{2\ell})$

1: for each vertex v in $V_{2\ell}$ constructs the following data structures:
$Array(Id(L(v),*))$, $List(Id(L(v),*))$, $Array(Id(*,R(v)))$, and $Array(Id(*,R(v)))$
2: for each edge $e = (v_i,v_j)$ in E_ℓ do
3: let $i = Id(v_i)$, $j = Id(v_j)$, and $c = w(e)$
4: for each k in $List(Id(j,*))$
5: if $Array(Id(i,*))[k]=1$
6: then construct edge between $Id(i, k)$ and $Id(j, k)$ in $G_{2\ell} = (V_{2\ell}, E_{2\ell})$ and let weight of this edge be c
7: for each k in $List(Id(*, j))$
8: if $Array(Id(*,i))[k]=1$
9: then construct edge between $Id(k, i)$ and $Id(k, i)$ in $G_{2\ell} = (V_{2\ell}, E_{2\ell})$ and let weight of this edge be $c + \ell$

演算法的第 1 個步驟針對 2ℓ 位元集合中每一個元素 v 建立如下四個資料結構：$Array(Id(L(v),*))$, $List(Id(L(v),*))$, $Array(Id(*,R(v)))$,與 $Array(Id(*,R(v)))$。建立的方式如下：令 $Id(L(X)) = i$ 且 $Id(R(X)) = j$。則將 $Array(Id(i,*))[j]$ 與 $Array(Id(*, j))[i]$ 皆設為 1,且將 i 與 j 分別放入 $List(Id(i,*))$ 與 $List(Id(*, j))$ 中。第 2 個步驟則是根據 ℓ 位元關係圖的邊，建立 2ℓ 位元關係圖的邊。若 $e = (v_i,v_j)$是 ℓ 位元關係圖的邊，則表示 v_i 與 v_j 是 ℓ 位元集合的元素且其 HD 為 1。因此對所有 2ℓ 位元集合的元素 X 與 Y，其中 $L(X) = v_i$ 且 $L(Y) = v_j$，只要 $R(X) = R(Y)$，則 X 與 Y 的 HD 就為 1。故演算法步驟 4 到 6 則是針對每一個 $List(Id(j,*))$ 中的元素，也就是所有 $L(Y) = j$ 的 Y，檢查 $R(Y)$是否等於 $R(X)$。同理，對所有 2ℓ 位元集合的元素 X 與 Y，其中 $R(X) = v_i$ 且 $R(Y) = v_j$，只要 $L(X) = L(Y)$，則 X 與 Y 的 HD 就為 1。演算法步驟 7 到 9 便是針對每一個 $List(Id(*, j))$ 中的元素，檢查 $L(Y)$是否等於 $L(X)$。

定理 3.1

演算法 3.1 根據 $G_\ell = (V_\ell, E_\ell)$ 成功建構 $G_{2\ell} = (V_{2\ell}, E_{2\ell})$。

證明:

由引理 3.1 知，若兩個 2ℓ 位元集合的元素 X 與 Y 之 $HD = 1$，則 X 與 Y 的關係必然是左半 $HD = 1$，右半相同，或者右半 $HD = 1$，左半相同。因為在 $Sp^{(*)}(S_{cs})$為命名編號表，ℓ 位元集合 SB_ℓ 是由 $SB_{2\ell}$ 分左右得到的，因此 $SB_\ell = SP(SB_{2\ell})$，亦即 SB_ℓ 是所有可能形成 $SB_{2\ell}$ 的 ℓ 位元字串的集合。演算法步驟 4 到 6 檢查了所有的左半 $HD = 1$、右半相同的情形，演算法步驟 7 到 9 檢查了所有右半 $HD = 1$、左半相同的情形。此外，考慮邊上的權重的給定。演算法步驟 4 到 6 檢查了所有的左半 $HD = 1$、右半相同的情形，因此 $Id(i, k)$與 $Id(j, k)$仍會在第 c 個位置上不同。而演算法步驟 7 到 9 檢查了所有右半 $HD = 1$、左半相同的情形。則 $Id(k, i)$ 與 $Id(k, i)$會在第 $c + \ell$ 個位置不同，因為在右半第 c 個位置不同，是全部的 $c + \ell$ 個位置不同。故演算法成功由 $G_\ell = (V_\ell, E_\ell)$ 建構 $G_{2\ell} = (V_{2\ell}, E_{2\ell})$。

■

3.3 時間複雜度

本節將分析 K=1 的近似共同區演算法的時間雜度。我們的演算法首先利用 [8]的方法找出字串集合上所有的字元集合，並加以編碼。我們由 1 位元關係圖得到 2 位元關係圖，再由 2 位元關係圖得到 4 位元關係圖，依序疊代下去，最後由 $n/2$ 位元關係圖得到 n 位元關係圖。演算法的時間分析如下。

定理 3.2

K=1 的近似共同區演算法共需 $O(n^3)$ 的時間。

證明：

首先利用[8]的方法找出字串集合上所有的字

元集合，並加以編碼。這個部分需要花 $O(n^2 \log n)$ 的時間。接著我們討論由 G_ℓ 建構 $G_{2\ell}$ 演算法所需要的時間。演算法的第 1 個步驟針對 2ℓ 位元集合中每一個元素 v 建立如下四個資料結構，這部分需花 $O(|V_{2\ell}|)$ 的時間，因為 $V_{2\ell}$ 中的每一個元素都只需常數步驟就可以完成資料結構的建構。接著考慮步驟 4 到 6 所需的時間。這個部分必須考慮所有 G_ℓ 的邊，也就是對每一個 V_ℓ 的點，我們必須考慮所有和其相鄰的邊。因為 V_ℓ 為長度 ℓ 的字串，因此和其 *HD* 為 1 的字串最多有 ℓ 個(最多只有 ℓ 個位置不同)。因此，每一個 $SB_{2\ell}$ 的元素在建構 $G_{2\ell}$ 最多需比較 ℓ 次。同理，步驟 7 到 9 也要花相同的時間。

因此，考慮 G_n 的建構，因為 $|V_n| = O(n^2)$ 且 $G_{n/2}$ 每個點最多與 *n*/2 個點相連。若將 G_n 的建構分成兩個部分，第一部分為資料結構之建構，此部分最多花 $O(n^2)$ 的時間。第二部分為 G_n 邊的建構，此部分最多需 $n^2(n/2)$的時間。由此類推 $G_{n/2}$ 的建構，則第一部份需花最多 $O(n^2)$ 的時間，第二部份最多花 $n^2(n/4)$的時間。

考慮所有建構 $G_\ell = (V_\ell, E_\ell)$, $\ell = 2, 4, 8, \ldots, 2^k = n$ 的時間總和。假設建構每一個圖的第一部份都花 $O(n^2)$ 的時間，則全部共花 $k \cdot n^2 = O(n^2 \log n)$ 的時間。第二個部分需要的時間總和為：

$n^2(n/2+n/4+n/8\ldots) = O(n^3)$

因此，演算法所需的時間最多為 $O(n^2 \log n) + O(n^3) = O(n^3)$ ■

4. *K*>1 的近似共同區演算法

當我們得 $G_n = (V_n, E_n)$ 之後，我們便知道了所有 *HD*=1的集合間的關係。如果我們要進一步瞭解對於某個特定集合 v，有哪些集合和它的 *HD*=*K*，則我們可以利用這樣簡單的方式。我們在 G_n 上以 v 為起點做深度優先搜尋，定出所有與 v 距離為 K 的集合。此時距離的計算必須和傳統深度優先的距離定義不同，也就是集合 u 與集合 v 之間的距離並非連結這兩個點的路徑的長度（邊的個數），而是在這條路徑上，有幾個字元在其路徑的邊上出現了奇數次。若一個字元 a 在這個路徑的邊上出現偶數次，則表示這個字元 a 先增加到路徑中某個集合，後來又從路徑中的某個集合拿掉。因此對原始集合 v 的而言，最後的集合並沒有相差 a 這個字元。我們可以利用一個 *couter* 陣列計算幾個字元在其路徑的邊上出現了奇數次。假設 $\sum = \{a_1, a_2, ..., a_{|\sum|}\}$，則一開始令所有的 *couter*[*i*] = 0。當經過一條值為 a_i 的邊時，則將 *couter*[*i*] 的內容01互換。此時所有的 *couter*[*i*] 的總和即是從 v 開始，到現在目前這個端點，其其路徑的邊上有幾個字元在出現了奇數次。若這個值是 K，則將輸出現在這個端點所對應的集合，表示這個集合和集合 v 的 *HD* 為 K。這個部分所需的時間即是深度優先搜尋所需的時間，其為 $O(|E_n|)$。

5. 結論

在本論文中，我們以近似共同區的方式討論基因群組的尋找問題，並提出一個 $O(n^3)$ 的演算法決定字串之間近似共同區間的關係。我們的方法使用命名法則的觀念，將比 Amir 等人所提的方法更為簡單。目前演算法的時間分析雖然得到的是 $O(n^3)$ 時間複雜度，但我們相信經過更嚴格的時間分析，演算法應該可以得 $O(n \log^2 n)$ 的時間複雜度。目前我們在正著手進行這部份的改進。此外我們也考慮以其他的定義來討論基因群組，使得基因群組的定義更合於實際的生物意義。

參考文獻

[1] Amir, A., Apostolico, A., Landau, G.M., Satta, G., “Efficient text fingerprinting via Parikh mapping”, ***J. Discrete Algorithms***, 26, 1–13, 2003.

[2] Amir, A., Gasieniec, L., Shalom, R.

"Improved approximate common interval". ***Information Processing Letters***, 103, 142-149, 2007.

[3] B'erard, S., Bergeron, A., Chauve, C. "Conserved structures in evolution scenarios". ***Lecture Notes in Bioinformatics*****,** 3388, 1–15, 2005.

[4] B'erard, S., Bergeron, A., Chauve, C., Paul, C.: "Perfect sorting by reversals is not always difficult". ***Lecture Notes in Bioinformatics***, 3692, 228–238, 2005.

[5] Bergeron, A., Blanchette, M., Chateau, A., Chauve, C. "Reconstructing ancestral gene orders using conserved intervals". ***Lecture Notes in Bioinformatics***, 3240, 14–25, 2004.

[6] Blin, G., Chateau, A., Chauve, C., Gingras, Y. "Inferring positional homologs with common intervals of sequences". ***Lecture Notes in Bioinformatics***, 4205, 24–38, 2006.

[7] Bourque, G., Yacef, Y. El-Mabrouk, N. "Maximizing synteny blocks to identify ancestral homologs". ***Lecture Notes in Bioinformatics***, 3678, 21–34, 2005.

[8] G. Didier. "Common intervals of two sequences". ***Proceedings of the Third International Workshop on Algorithms in Bioinformatics,*** WABI 2003,17-24, 2003.

[9] Hoberman, R., Durand, D. "The incompatible desiderata of gene cluster properties". ***Lecture Notes in Bioinformatics***, 3678, 73–87, 2005.

[10] Semon, M., Duret, L. "Evolutionary origin and maintenance of coexpressed gene clusters in mammals". ***Molecular Biology and Evolution***, 23(9), 1715–1723, 2006.

[11] T. Schmidt, J. Stoye, "Quadratic time algorithms for finding common intervals in two and more sequences", ***Proc. 15th Annual Symposium on Combinatorial Pattern Matching, in: Lecture Notes in Comput. Sci***, 3109, 347–358, 2004.

[12] T. Uno and M. Yagiura. "Fast algorithms to enumerate all common intervals of two permutations". ***Algorithmica***, 26, 290-309, 2000.

[13] I. Yanai, C. DeLisi, "The society of genes: Networks of functional links between genes from comparative genomics", ***Genome Biol***, 3 (64), 1–12, 2002.