

Sequence Comparison

①

Definition (Alignment)

An alignment of two sequences S_1 and S_2 is a pair of sequences (S'_1, S'_2) obtained by insertion of spaces in S_1 and S_2 (respectively) such that

(1) $|S'_1| = |S'_2|$, and

(2) $\forall i, S'_1(i)$ is aligned with $S'_2(i)$ and either $S'_1(i)$ or $S'_2(i)$ is not a space.

(Note) S_1 can be viewed as a subsequence of S'_1 if (S_2) (S'_2) if we consider the insertions as elements.

Example

$$S_1 = \text{AGAC}$$

$$S_2 = \text{TACCC}$$

$$\begin{array}{l|l} S'_1 = *AGAC & \text{or} \\ S'_2 = TACCC & \begin{array}{l} S'_1 = *AGAC* \\ \downarrow \quad \downarrow \downarrow \\ S'_2 = \text{TA}CC* C \end{array} \end{array}$$

Score of Alignment

$$a(x, y) = \begin{cases} p & \geq 0 & \text{if } x = y \\ q & < 0 & \text{if } x \neq y \text{ and } y \text{ is a " " } \\ r & < 0 & \text{if one of } x, y \text{ is a " " } \end{cases}$$

Similarity of S_1 and S_2

$$|S_1| = |S_2|$$

$$Sim(S_1, S_2) = \max_{\substack{(S'_1, S'_2) \\ \text{all alignments}}} \left\{ \alpha(S'_1, S'_2) = \sum_{i=1}^k \alpha(S'_1(i), S'_2(i)) \right\}.$$

Definition (Optimal Alignment)

The pair (S'_1, S'_2) with largest score is called an optimal alignment.

For example: (For AGC and AAAC)

* AGC
A A A C

A * G C
A A A C

A G * C
A A A C

are optimal alignments with $Sim(AGC, AAAC) = -1$.

$$p = 1, q = -2$$

Dynamic Algorithm

" $i, j \in \{1, 2, \dots, n\}$ "

$$a[i, j] = Sim(S_1[0, i], S_2[0, j])$$

$$= \max \begin{cases} a[i, j-1] + q \\ a[i-1, j-1] + p \\ a[i-1, j] + q \end{cases}$$

$$a[i, 0] = i \cdot q$$

$$a[0, j] = j \cdot q$$

	*	A	G	C
*	0 → -2 → -4 → -6			
A	-2	1 → -1 → -3		
A	-4	-1	0 → -2	
A	-6	-3	-2	-1
C	-8	-5	-4	-1

3

Theorem The time to accomplish "finding optimal alignment" is $O(|S_1| \cdot |S_2|)$ and the space we need is also $O(|S_1| \cdot |S_2|)$.

Proof. Since constant work is required per entry in the matrix, the proof follows. ■

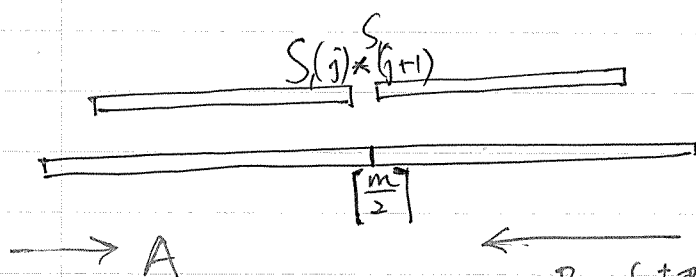
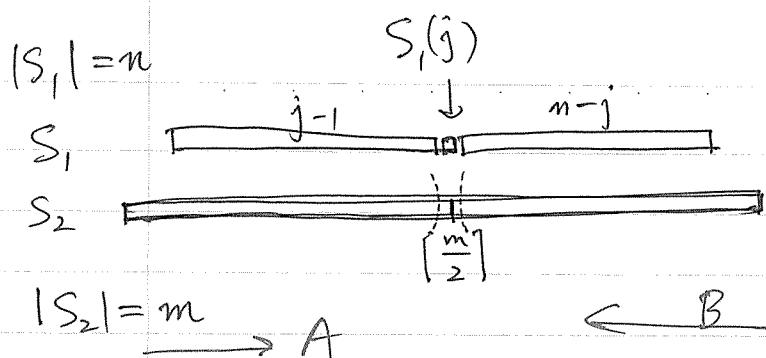
Note No algorithm is known that uses asymptotically less time and has the same generality. Although, there are algorithms for more specific problems.

Note 儲存兩個相對的 sequences 並不難; 但是, 如果每個 sequence 都有 10,000 字元, 則寫出矩陣就要填入 $(10,000)^2 = 100,000,000$, 壹億個數, 是龐大的空間。

(*) We are aiming to find the "optimal alignments" instead of "presenting the matrix".

Another idea (By Bonnie Berger)

④



B (把两 sequences 都倒过来看)

Step 1 選擇中間的字母 (從 S_2 中)。

(這個字母所對的位置不外乎是 $S_1(j)$ 或 $S_1(j) \times S_1(j+1)$ 中間的空格。)

Step 2 計算 ($\forall j$)

$$\max \begin{cases} (A(\lceil \frac{m}{2} \rceil - 1, j - 1) \pm 1) + B(\lfloor \frac{m}{2} \rfloor, n - j); \text{ and} \\ (A(\lceil \frac{m}{2} \rceil - 1, j) - 2) + B(\lfloor \frac{m}{2} \rfloor, n - j). \end{cases}$$

Step 3

The maximum tell you the best score for aligning the $\frac{m}{2}$ (th) character of S_2 with some character of S_1 or gap. ($2n$ values!)

(5)

Time we need

$$T(m, n) = 2cmn$$

$$T(m, 1) = m$$

$$T(1, n) = n$$

$$T(m, n) \leq cmn + \max_{j=1}^n \left(2c \left(\left\lceil \frac{m}{2} \right\rceil - 1 \right) \cdot j + 2c \left\lfloor \frac{m}{2} \right\rfloor (n-j) \right)$$

By induction

$$\leq cmn + \max_{j=1}^n (cmj + cm(n-j))$$

$$\leq cmn + cmn$$

$$\leq 2cmn$$

↑ Global AlignmentsSemi-global Comparison :

To find the optimal alignment between suffix of S_1
 (prefix)
 with prefix of S_2
 (suffix)

Example S_1 : CAGCA*CTTGGATTCTCGG

S_2 : CAGCGTGG

(6)

Multiple Sequence Alignment (MSA)

Definition (Sum-of-pairs, SP-score)

Let α be a collection of alignments of $S_1, S_2, \dots, S_n$:
 $S'_1, S'_2, \dots, S'_n$. Then, the SP-score of α , denoted by
 $SP(\alpha) = \sum_{1 \leq i < j \leq n} \sigma(S'_i, S'_j)$, $\sigma(S'_i, S'_j)$ is called the score
 of $\alpha_{i,j}$, denoted by $s(\alpha_{i,j})$

(*) Since MSA's are applied in finding the similarity of protein sequences, we use characters of proteins for examples.

S_1 : RCTLEE

S'_1 : RCTLEE

S_2 : RCLEE

$\xRightarrow{\alpha} S'_2$: RC * LEE

S_3 : CTLEE

S'_3 : * CTLEE

S_4 : CTEE

S'_4 : * CT * EE

(With $p=1$ and $q=-2$)

$* \leftrightarrow *$
score 0

$$SP(\alpha) = s(\alpha_{1,2}) + s(\alpha_{1,3}) + s(\alpha_{1,4}) + s(\alpha_{2,3}) + s(\alpha_{2,4}) + s(\alpha_{3,4})$$

$$= 3 + 3 + 0 + 0 + (-3) + 2 = 5.$$

6

[Sequence Alignment
Terms Explained](#)[Sequence Alignment
Software](#)[Sequence Alignment Web
Resources](#)[Glossary](#)[About](#)[Contact](#)

Sequence Alignment Terms

Alignment

The comparison of two or more nucleotide or protein sequences to determine the degree of similarity. Commonly used to deduct functional or evolutionary relationships between genes and proteins.

Assembly

The process of combining short DNA sequence fragments into larger units by looking for overlaps between different fragments. Often required because the length of the genes studied exceeds the length of the sequence fragments produced by DNA sequencing machines. Also used to combine several fragments that cover the same region, for example in forward and reverse direction, with the goal to reduce errors in the consensus sequence.

Consensus Sequence

A single sequence generated from an alignment or assembly of sequence fragments that is the "best fit" for the given sequences. Historically, majority ("vote based") and inclusive methods were most commonly used to determine consensus sequence. For sequence assemblies, these methods have often been replaced by quality-based consensus methods. Quality-based consensus sequences are typically more accurate than majority-based sequences, and can reduce the need for manual editing of sequence assemblies drastically.

Contig

The result of a sequence assembly or alignment that shows the arrangement of the fragments to form a contiguous large sequence.

Dynamic Programming

A computer-science based [method](#) to find the optimal alignment between sequences. For two sequences, this algorithm creates a two-dimensional matrix based on identity or similarity of bases (or amino acids) in both sequences, and then finds the highest-scoring path to obtain the alignment. A commonly used dynamic programming method is the [Needleman-Wunsch algorithm](#). A nice graphical display of the dynamic programming methods for sequence alignments can be found [here](#).

Global Alignments

Global alignments attempt to align every base (or amino acid) in each aligned sequence.

Local Alignments

Local alignments will align only similar regions between sequences, and leave regions with too many differences unaligned. Local alignments can be better suited for the alignment of very dissimilar sequences. In sequence assembly, the program [Phrap](#) demonstrate that local alignments can be used to reduce or eliminate the need to remove low-quality sequence (end clipping) before assembly.

Multiple Sequence Alignment

[Multiple sequence alignment](#) refers to the process of aligning three or more nucleotide or protein sequences to identify similarities between the sequences. Alignments that include many sequences can be computationally intensive, and require more sophisticated algorithms than pairwise alignments.

Pairwise Alignment

In pairwise sequence alignment, exactly two nucleotide or protein sequences are aligned to each other to determine the similarity between the two sequences.

Word-based Alignment Methods

Word-based alignment methods are an optimization often used in sequence alignment and assemblies. Instead of examining every single nucleotide or amino acid, "words" of a fixed length are analyzed. This can lead to substantial reductions in memory use and alignment times. One common application is to use the number of shared words between two sequences to estimate the similarity in early phases of sequence alignments, or to identify sequences that share overlaps in sequence assembly.

②

MSA problem

Given k sequences, find the optimal alignment of these k sequences.

Dynamic Programming

Time-complexity = $O(k^2 2^k n^k)$

(Exponential in k !)

觀察: 每一次調整都要和其它的 Sequences 配一次!

研究進展

1. MSA problem is NP-complete

Wang and Jiang (1994) + Bonizzoni and Vedova (2001)

2. Gusfield (1993) : $(2 - \frac{2}{k})$ -approximation algorithm
(If $k=2$, then the algorithm is working.)

3. Pevzner (1992) : $(2 - \frac{3}{k})$ -approx. algorithm.

(*) 4. Bafna, Lawler and Pevzner (1997) : Approximation algorithms for MSA, Theoretical Computer Science, 182, 233-244.
 $(2 - \frac{l}{k})$ -approx. algorithm for $l < k$.

→ You may choose this paper for writing your report!

⑦'

[Home](#)[Sequence Alignment Software](#)[Sequence Alignment Web Resources](#)[Glossary](#)[About](#)[Contact](#)

DNA Sequence Alignment

Sequence alignment describes the way of aligning DNA, RNA, or protein sequences to highlight or identify similarities between DNA sequences. Typically, gaps have to be inserted into sequences so that identical or similar nucleotides or amino acids are aligned in columns. Here is an example:

Scarites	C	T	T	A	G	A	T	C	G	T	A	C	C	A	A	-	-	-	A	A	T	A	T	T	A	C
Carenum	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	A	-	T	A	C	-	T	T	T	A	C
Pasimachus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	T	A	T	A	A	G	T	T	T	A	C
Pheropsophus	C	T	T	A	G	A	T	C	G	T	T	C	C	A	C	-	-	-	A	C	A	T	A	T	A	C
Brachinus armiger	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	T	C
Brachinus hirsutus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	A	C
Aptinus	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	C	A	T	T	A	C	
Pseudomorpha	C	T	T	A	G	A	T	C	G	T	A	C	C	-	-	-	-	-	A	C	A	A	T	A	C	

Sequence alignment is often used in several biomedical research fields, including phylogenetics, biogeography, and evolution research.

Here is an example of a phylogenetics experiment that includes DNA sequence alignments. Our researcher works for a museum of natural history, and wants to study the evolutionary relation between several animal species he studies. A typical approach would be:

- Isolate DNA from each of the species studied
- Generate DNA sequences for a gene region, an entire gene, or several genes
- Assemble the DNA sequences for each species into separate "contigs"
- Edit the sequence assembly results to remove any errors
- Align the contig sequences to each other using a multiple sequence alignment program
- Verify any observed differences by going back to the original DNA sequences
- Correct the placement of gaps in the aligned sequences, if necessary
- Export the sequence alignment for further analysis with phylogenetics software, for example to generate phylogenetic trees

This web site provides links to commonly used programs and web resources for DNA sequence alignments. Since hundreds of different programs and relevant web sites exist, the goal is *not* to provide lists, but rather to concentrate on the most commonly used and the most useful sequence alignment software.

Scoring

(*) 给分的目的在于判断 sequences 之间的相似程度, 亦即如何"修改"比较可能; 对于相同或相似的字母比对, 给分相对一致, 亦即 $\pm p$, 然而, 与空格的比对, 变化很多:

(i) 空格对空格,

(ii) 连续空格的作用与断续空格的作用不同。

(前者受生几率大, 尤其是在 Local 的比对时, 后者 (Semiglobal) 受生; 而后者一般而言比较不会受生, 尤其是多个分散开来的空格, 这样的 Alignment 意义不大。

(**) 简单的分数给法, 通常不能完全反映真实蛋白质的变异演化特性。But, how? $\searrow$

(***)(topic) A good question to study.

How about spaces?

	A	C	G	T
A	p	a	c	d
C	a	p	e	f
G	c	e	p	b
T	d	f	b	p

Scoring Systems

Simplest

$$\delta(x, x) = 1, \delta(x, y) = -\mu \quad (x \neq y) \text{ and } \delta(x, *) = \delta(*, y) = -c$$

↓

	A	T	G	C
A	-	p	g	n
T	p	-	a	t
G	g	a	-	u
C	n	t	u	-

p, g, n, a, t, u are negative scores

$x_1 \ x_2 \ \dots \ x_t$

$* \ * \ \dots \ *$

$-t \alpha + \boxed{}$

positive number

$f(t)$

$$= \begin{cases} 0 & \text{if } t=1 \\ - & \text{ } \end{cases}$$

1966, Levenshtein

Definition (Edit Distance)

The edit distance between two strings (sequences) is the minimum number of edit operations need to transform one string into another, where the edit operations are (1) (sequence) insertion of a symbol (character), (2) deletion of a symbol, and (3) substitution of a symbol for another one.

(All known algorithms are using dynamic programming.)

(*) Finding differences (edit distance) between sequences is often equivalent to finding similarities between two sequences.

(**) If edit operations are (1) and (2) only, then the edit distance problem is equivalent to the "Longest Common Subsequence (LCS)" problem.

(***) Studies of the symmetric group revealed surprising connections between representation theory and the problem of finding the LCS between two permutations.

(离散数学)  (viewed as sequences)

T G C A T A
A T C T G A T

Common subsequence
of length "4"

W: (* T G C A T * A *)
V: (A T * C * T G A T)

Edit distance
"5"

From V to W: 2 insertions and 3 deletions.

=

Longest Common Subsequence Problem.

Definition (Common Subsequence)

Let $V = \langle v_1, v_2, \dots, v_n \rangle$ and $W = \langle w_1, w_2, \dots, w_m \rangle$. A common
subsequence of V and W is a sequence
of indices $1 \leq i_1 \leq i_2 \leq \dots \leq i_k \leq n$, and a sequence
chosen from V
of indices $1 \leq j_1 \leq j_2 \leq \dots \leq j_k \leq m$ chosen from W such that

$$v_{i_k} = w_{j_k} \text{ for } k = 1, 2, \dots, k.$$

For example

$V = \text{ATCTGAT}$

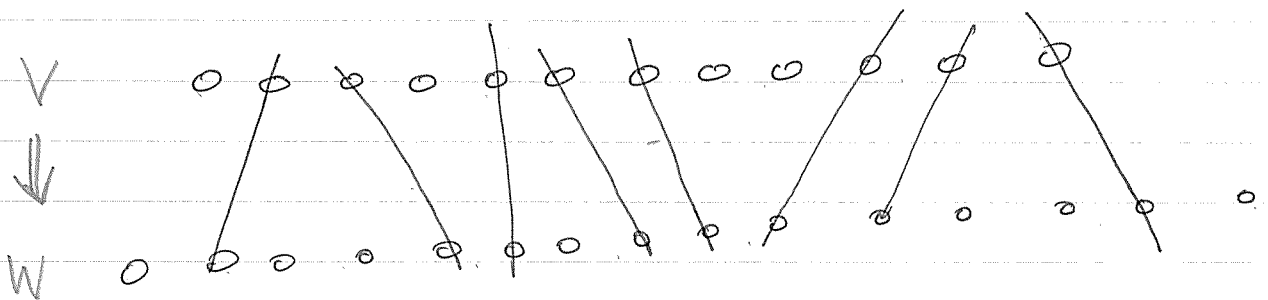
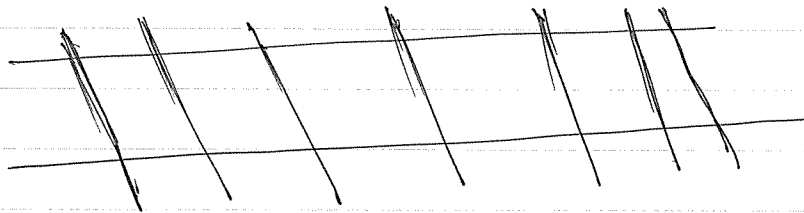
$W = \text{TGCATA}$

$$\left\{ \begin{array}{l} v_2 = v_{i_1}, v_3 = v_{i_2}, v_4 = v_{i_3}, v_6 = v_{i_4} \\ w_1 = w_{j_1}, w_3 = w_{j_2}, w_5 = w_{j_3}, w_6 = w_{j_4} \end{array} \right.$$

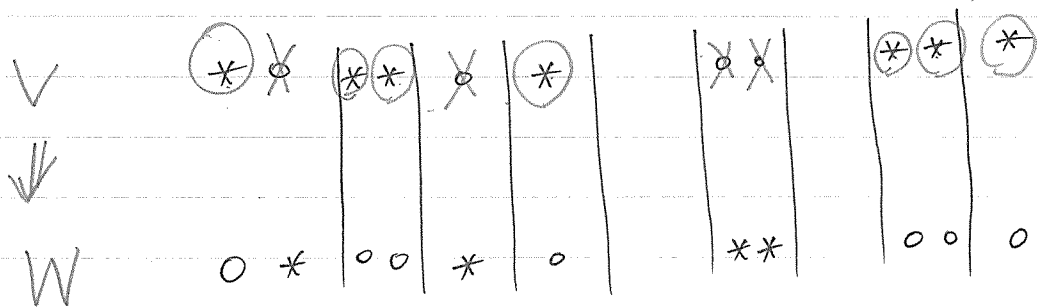
11

Let $s(V, W)$ be the length (k) of a largest common subsequence (LCS) of V and W . Clearly, we have

(Fact 1) $d(V, W) = n + m - 2s(V, W)$ is the minimum number of insertions and deletions needed to transform from V into W .



$$11 = 12 + 15 - 2 \cdot (8)$$



Compute $s(V, W)$, $V = \langle v_1 v_2 \dots v_n \rangle$ and $W = \langle w_1 w_2 \dots w_m \rangle$

Algorithm (Dynamic Algorithm)

Define $s_{i,j} = s(V_i, W_j)$ where $V_i = \langle v_1 v_2 \dots v_i \rangle$ and

$W_j = \langle w_1 w_2 \dots w_j \rangle$, $s_{i,0} = s_{0,j} = 0$, $\forall 1 \leq i \leq n$ and $1 \leq j \leq m$.

Step 1

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j. \end{cases}$$

	*	T	G	C	A	T	A	
		← W						
*		0 ₀	0 ₁	0 ₂	0 ₃	0 ₄	0 ₅	0 ₆
A		0 ₁	0 ₂	0 ₃	0 ₄	1 ₃	1 ₄	1 ₅
T		0 ₂	1 ₁	1 ₂	1 ₃	1 ₄	2 ₃	2 ₄
C		0 ₃	1 ₂	1 ₃	2 ₂	2 ₃	2 ₄	2 ₅
T		0 ₄	1 ₃	1 ₄	2 ₃	2 ₄	3 ₃	3 ₄
G		0 ₅	1 ₄	2 ₃	2 ₄	2 ₅	3 ₄	3 ₅
A		0 ₆	1 ₅	2 ₄	2 ₅	3 ₄	3 ₅	4 ₄
T		0 ₇	1 ₆	2 ₅	2 ₆	3 ₅	4 ₄	4 ₅
	↑ V							

$$s(V, W) = 4 \quad d(V, W) = 5 \text{ (indices)}$$

$$\checkmark d(V, W) = 7 + 6 - 2(4) = 5. \quad \text{By (Fact 1).}$$

Compute $d(V, W)$

Algorithm (Dynamic)

$$d_{i,0} = i$$

$$d_{0,j} = j$$

$$d_{i,j} = \min. \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} \text{ if } v_i = w_j. \end{cases}$$

Example See page (12).

Application

Let $\pi = (x_1 x_2 \dots x_n)$ be a permutation in S_n .

Finding a ^{of π} longest increasing subsequence is equivalent to find the LCS between π and the permutation $(1 2 3 \dots n)$.

(*) It is well-known that every permutation on n elements has either an increasing or a decreasing subsequence of length at least $\sqrt{n}$.