Pairwise Sequence Alignment

Solon P. Pissis

CWI & VU, Amsterdam

October 31, 2024

Contents

Introduction

- Sequence alignment is the process of comparing two or more strings of letters (e.g. nucleotides or amino acids) to:
  - infer their similarity; or
  - visualize their similarity.
- Pairwise sequence alignment is the comparison of two strings.
- Useful in dozens of biological applications; e.g.
  - genome assembly; or
  - phylogeny reconstruction.

	0	1	2	3	4	5	6	7	8
$x =$	G	C	G	A	C	G	T	C	C
	|	|	|	|			|	.	|
$y =$	G	C	G	A	−	−	T	A	C

Figure: Alignment of $x =$ GCGACGTCC and $y =$ GCGATAC: one mismatch at position 7 and a gap of length two inserted in y after position 3

Introduction

- We focus on online sequence alignment — the sequences cannot be preprocessed to build an index on them.
- There exist four main approaches to online sequence alignment:
  - **dynamic programming** (DP) based algorithms;
  - automata-based algorithms;
  - bit-parallel algorithms;
  - filtering-based algorithms.
- There mainly exist two different distances for comparing two strings:
  - **edit distance** [1, 3];
  - Hamming distance.
- Biological applications require the modification of algorithms measuring the distance between two strings to perform:
  - **global** alignment;
  - **local** alignment.

Introduction

C	G	T	C	C	G	A	A	G	T	G
		|	.	|	|	|	|			
–	–	T	A	C	G	A	A	–	–	–

Table: Global alignment between $x =$ CGTCCGAAGTG and $y =$ TACGAA

T	C	C	G	A	A
|	.	|	|	|	|
T	A	C	G	A	A

Table: Local alignment between $x =$ CGTCCGAAGTG and $y =$ TACGAA

Contents

Alphabet and strings

Definition (Alphabet)

An *alphabet* Σ is a set whose elements are called *letters*.

Definition (String)

A *string* on Σ is a sequence of elements of Σ.

The zero-letter sequence ε is called the *empty string*.
The set of all possible strings on the alphabet Σ is denoted by Σ^*.

Definition (Length of string)

The *length* of a string x is the length $|x|$ of the sequence.

Alphabet and strings

We denote by $x[i]$, for all $0 \leq i < |x|$, the letter at index i of x. We also call index i, for all $0 \leq i < |x|$, a position in x. It follows that the ith letter of x is the letter at position $i - 1$ in x, and that

$$x = x[0\,..\,|x| - 1]$$

Definition (Identity between strings)

The *identity* between any two strings x and y is defined as

$$x = y$$

if and only if

$$|x| = |y| \text{ and } x[i] = y[i], \text{for all } 0 \leq i < |x|$$

Alphabet and strings

Definition (Concatenation of strings)

The *concatenation* of two strings x and y is the string of the letters of x followed by the letters of y. It is denoted by xy.

Definition (Substring of string)

A string x is a *substring* of a string y if there exist two strings u and v, such that $y = uxv$.

Notice that u and v are possibly empty strings!

Definition (Occurrence of string)

Let x be a non-empty string and y be a string. We say that there exists an *occurrence* of x in y, or, more simply, that x *occurs in* y, when x is a substring of y.

Distance

Definition (Distance between two strings)

We say that a function $\delta : \Sigma^* \times \Sigma^* \to \mathbb{R}$ is a *distance* on Σ^* if the four following properties are satisfied, for every $u, v \in \Sigma^*$:

- *Positivity*: $\delta(u, v) \geq 0$
- *Separation*: $\delta(u, v) = 0$ if and only if $u = v$
- *Symmetry*: $\delta(u, v) = \delta(v, u)$
- *Triangle inequality*: $\delta(u, v) \leq \delta(u, w) + \delta(w, v)$, for $w \in \Sigma^*$

The distances are defined from operations that transform x into y. Three types of elementary operations are considered.

- *substitution* (sub) for a letter of x by a letter of y
- *deletion* (del) of a letter of x
- *insertion* (ins) of a letter of y in x

Edit distance

We assume $\mathrm{sub}(a, b) := \mathrm{sub}(b, a) := \mathrm{del}(a) := \mathrm{ins}(b) := 1$.

Definition (Edit distance)

From the elementary costs, we set

$$\delta_E(x, y) = \min\{\text{cost of } \sigma : \sigma \in S_{x,y}\}$$

where $S_{x,y}$ is the set of sequences of elementary edit operations that transform x into y, and the cost of an element $\sigma \in S_{x,y}$ is the sum of the costs of the edit operations of the sequence σ. The function δ_E is then a distance on Σ^* called the *edit distance*.

Hamming distance

Definition (Hamming distance)

The *Hamming distance*, denoted by $\delta_H(x, y)$, is defined for two strings x and y of the same length as the number of positions in which x and y possess different letters.

Alignment

Definition (Alignment between two strings)

An alignment between x and y is a string z on the alphabet of pairs of letters, more accurately on

$$(\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \setminus (\{\varepsilon, \varepsilon\})$$

whose projection on the first component is x, and the projection on the second component is y. Thus, if z is an alignment of length p between x and y, we have

$$z = (x'_0, y'_0)(x'_1, y'_1) \dots (x'_{p-1}, y'_{p-1})$$

$$x = x'_0 x'_1 \dots x'_{p-1}$$

$$y = y'_0 y'_1 \dots y'_{p-1}$$

where $x'_i \in \Sigma \cup \{\varepsilon\}$ and $y'_i \in \Sigma \cup \{\varepsilon\}$, for all $0 \leq i < p$.

Example

Example

Let the string $x = \texttt{ACGA}$ and the string $y = \texttt{ATGCTA}$.
An alignment between x and y is

$$\begin{pmatrix} \texttt{ACG--A} \\ \texttt{ATGCTA} \end{pmatrix}$$

Operation	Aligned pair	Cost
substitute A for A	(A,A)	0
substitute T for C	(C,T)	1
substitute G for G	(G,G)	0
insert C	(-,C)	1
insert T	(-,T)	1
substitute A for A	(A,A)	0

This alignment is optimal since its cost is $\delta_E = 3$.

Contents

Edit distance

We focus on algorithms based on Dynamic Programming (DP).

Let x and y be two strings of lengths m and n, respectively.
We will assume a 1-based index only for these two strings:

$$x = x[1\,..\,m] \qquad\qquad y = y[1\,..\,n].$$

The cells of the DP matrix $T[0\,..\,m, 0\,..\,n]$ can be computed by:

$$T[0,0] = 0,$$

$$T[i,0] = T[i-1,0] + \mathsf{del}(x[i]),$$

$$T[0,j] = T[0,j-1] + \mathsf{ins}(y[j]),$$

$$T[i,j] = \min \begin{cases} T[i-1,j-1] + \mathsf{sub}(x[i], y[j]) \\ T[i-1,j] + \mathsf{del}(x[i]) \\ T[i,j-1] + \mathsf{ins}(y[j]). \end{cases}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

Edit distance - Example 1

Let $x = \texttt{EAWACQGKL}$, $y = \texttt{ERDAWCQPGKWY}$, $\text{sub}(a, b) := 3$, $\text{ins}(a) := 1$, and $\text{del}(a) := 1$, where Σ is the amino acids alphabet.

$$T[0,0] = 0,$$

$$T[i,0] = T[i-1,0] + \text{del}(x[i]),$$

$$T[0,j] = T[0,j-1] + \text{ins}(y[j]),$$

$$T[i,j] = \min \begin{cases} T[i-1,j-1] + \text{sub}(x[i], y[j]) \\ T[i-1,j] + \text{del}(x[i]) \\ T[i,j-1] + \text{ins}(y[j]). \end{cases}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

		0	1	2	3	4	5	6	7	8	9	10	11	12
	T		E	R	D	A	W	C	Q	P	G	K	W	Y
0		0	1	2	3	4	5	6	7	8	9	10	11	12
1	E	1												
2	A	2												
3	W	3												
4	A	4												
5	C	5												
6	Q	6												
7	G	7												
8	K	8												
9	L	9												

Edit distance - Example 1

Let $x = \texttt{EAWACQGKL}$, $y = \texttt{ERDAWCQPGKWY}$, $\text{sub}(a, b) := 3$, $\text{ins}(a) := 1$, and $\text{del}(a) := 1$, where Σ is the amino acids alphabet.

$$T[0,0] = 0,$$

$$T[i,0] = T[i-1,0] + \text{del}(x[i]),$$

$$T[0,j] = T[0,j-1] + \text{ins}(y[j]),$$

$$T[i,j] = \min \begin{cases} T[i-1,j-1] + \text{sub}(x[i], y[j]) \\ T[i-1,j] + \text{del}(x[i]) \\ T[i,j-1] + \text{ins}(y[j]). \end{cases}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

		0	1	2	3	4	5	6	7	8	9	10	11	12
	T		E	R	D	A	W	C	Q	P	G	K	W	Y
0		0	1	2	3	4	5	6	7	8	9	10	11	12
1	E	1	0											
2	A	2												
3	W	3												
4	A	4												
5	C	5												
6	Q	6												
7	G	7												
8	K	8												
9	L	9												

Edit distance - Example 1

Let $x = \texttt{EAWACQGKL}$, $y = \texttt{ERDAWCQPGKWY}$, $\text{sub}(a, b) := 3$, $\text{ins}(a) := 1$, and $\text{del}(a) := 1$, where Σ is the amino acids alphabet.

$$T[0,0] = 0,$$

$$T[i,0] = T[i-1,0] + \text{del}(x[i]),$$

$$T[0,j] = T[0,j-1] + \text{ins}(y[j]),$$

$$T[i,j] = \min \begin{cases} T[i-1,j-1] + \text{sub}(x[i], y[j]) \\ T[i-1,j] + \text{del}(x[i]) \\ T[i,j-1] + \text{ins}(y[j]). \end{cases}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

		0	1	2	3	4	5	6	7	8	9	10	11	12
	T		E	R	D	A	W	C	Q	P	G	K	W	Y
0		0	1	2	3	4	5	6	7	8	9	10	11	12
1	E	1	0	1	2	3	4	5	6	7	8	9	10	11
2	A	2	1	2	3	2	3	4	5	6	7	8	9	10
3	W	3	2	3	4	3	2	3	4	5	6	7	8	9
4	A	4	3	4	5	4	3	4	5	6	7	8	9	10
5	C	5	4	5	6	5	4	3	4	5	6	7	8	9
6	Q	6	5	6	7	6	5	4	3	4	5	6	7	8
7	G	7	6	7	8	7	6	5	4	5	4	5	6	7
8	K	8	7	8	9	8	7	6	5	6	5	4	5	6
9	L	9	8	9	10	9	8	7	6	7	6	5	6	7

Edit distance - Traceback

- How is the associated optimal trace extracted?
- Add pointers in the matrix as the values are computed.
- When $T[i,j]$ is computed set a pointer from cell $[i,j]$ to
  - cell $[i-1,j]$ if $T[i,j] = T[i-1,j] + \text{del}(x[i])$
  - cell $[i,j-1]$ if $T[i,j] = T[i,j-1] + \text{ins}(y[j])$
  - cell $[i-1,j-1]$ if $T[i,j] = T[i-1,j-1] + \text{sub}(x[i], y[j])$
- It is possible to point to **more than one cell**: we need to place a pointer to every cell which gives the minimum value.
- This rule applies to cells in row zero and column zero as well: each cell in row zero points to the cell to its left, and each cell in column zero points to the cell just above it.

Edit distance - Example 1 (Traceback)

T	-	E	R	D	A	W	C	Q	P	G	K	W	Y
-	0	1	2	3	4	5	6	7	8	9	10	11	12
E	1	0	1	2	3	4	5	6	7	8	9	10	11
A	2	1	2	3	2	3	4	5	6	7	8	9	10
W	3	2	3	4	3	2	3	4	5	6	7	8	9
A	4	3	4	5	4	3	4	5	6	7	8	9	10
C	5	4	5	6	5	4	3	4	5	6	7	8	9
Q	6	5	6	7	6	5	4	3	4	5	6	7	8
G	7	6	7	8	7	6	5	4	5	4	5	6	7
K	8	7	8	9	8	7	6	5	6	5	4	5	6
L	9	8	9	10	9	8	7	6	7	6	5	6	7

$$\begin{pmatrix} \texttt{E--AWACQ-GK--L} \\ \texttt{ERDAW-CQPGKWY-} \end{pmatrix} \begin{pmatrix} \texttt{E--AWACQ-GK-L-} \\ \texttt{ERDAW-CQPGKW-Y} \end{pmatrix} \begin{pmatrix} \texttt{E--AWACQ-GKL--} \\ \texttt{ERDAW-CQPGK-WY} \end{pmatrix}$$

Edit distance - Example 1 (Traceback)

T	-	E	R	D	A	W	C	Q	P	G	K	W	Y
-	0	1	2	3	4	5	6	7	8	9	10	11	12
E	1	0	1	2	3	4	5	6	7	8	9	10	11
A	2	1	2	3	2	3	4	5	6	7	8	9	10
W	3	2	3	4	3	2	3	4	5	6	7	8	9
A	4	3	4	5	4	3	4	5	6	7	8	9	10
C	5	4	5	6	5	4	3	4	5	6	7	8	9
Q	6	5	6	7	6	5	4	3	4	5	6	7	8
G	7	6	7	8	7	6	5	4	5	4	5	6	7
K	8	7	8	9	8	7	6	5	6	5	4	5	6
L	9	8	9	10	9	8	7	6	7	6	5	6	7

$$\begin{pmatrix}\texttt{E--AWACQ-GK--L}\\ \texttt{ERDAW-CQPGKWY-}\end{pmatrix}\begin{pmatrix}\texttt{E--AWACQ-GK-L-}\\ \texttt{ERDAW-CQPGKW-Y}\end{pmatrix}\begin{pmatrix}\texttt{E--AWACQ-GKL--}\\ \texttt{ERDAW-CQPGK-WY}\end{pmatrix}$$

Edit distance - Example 1 (Traceback)

T	-	E	R	D	A	W	C	Q	P	G	K	W	Y
-	0	1	2	3	4	5	6	7	8	9	10	11	12
E	1	0	1	2	3	4	5	6	7	8	9	10	11
A	2	1	2	3	2	3	4	5	6	7	8	9	10
W	3	2	3	4	3	2	3	4	5	6	7	8	9
A	4	3	4	5	4	3	4	5	6	7	8	9	10
C	5	4	5	6	5	4	3	4	5	6	7	8	9
Q	6	5	6	7	6	5	4	3	4	5	6	7	8
G	7	6	7	8	7	6	5	4	5	4	5	6	7
K	8	7	8	9	8	7	6	5	6	5	4	5	6
L	9	8	9	10	9	8	7	6	7	6	5	6	7

$$\begin{pmatrix}\texttt{E--AWACQ-GK--L}\\ \texttt{ERDAW-CQPGKWY-}\end{pmatrix}\begin{pmatrix}\texttt{E--AWACQ-GK-L-}\\ \texttt{ERDAW-CQPGKW-Y}\end{pmatrix}\begin{pmatrix}\texttt{E--AWACQ-GKL--}\\ \texttt{ERDAW-CQPGK-WY}\end{pmatrix}$$

Edit distance - Example 2

Let $x = \texttt{ACGA}$, $y = \texttt{ATGCTA}$, $\mathrm{sub}(a, b) := 1$, $\mathrm{ins}(a) := 1$, and $\mathrm{del}(a) := 1$, where Σ is the DNA alphabet.

$$T[0,0] = 0,$$

$$T[i,0] = T[i-1,0] + \mathrm{del}(x[i]),$$

$$T[0,j] = T[0,j-1] + \mathrm{ins}(y[j]),$$

$$T[i,j] = \min \begin{cases} T[i-1,j-1] + \mathrm{sub}(x[i], y[j]) \\ T[i-1,j] + \mathrm{del}(x[i]) \\ T[i,j-1] + \mathrm{ins}(y[j]). \end{cases}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

		0	1	2	3	4	5	6
	T		A	T	G	C	T	A
0		0	1	2	3	4	5	6
1	A	1	0	1	2	3	4	5
2	C	2	1	1	2	2	3	4
3	G	3	2	2	1	2	3	4
4	A	4	3	3	2	2	3	3

Edit distance - Example 2 (Traceback)

T	-	A	T	G	C	T	A
-	0	1	2	3	4	5	6
A	1	0	1	2	3	4	5
C	2	1	1	2	2	3	4
G	3	2	2	1	2	3	4
A	4	3	3	2	2	3	3

$$\begin{pmatrix} \texttt{A--CGA} \\ \texttt{ATGCTA} \end{pmatrix} \begin{pmatrix} \texttt{ACG--A} \\ \texttt{ATGCTA} \end{pmatrix}$$

Edit distance - Complexities

- This DP algorithm has been rediscovered many times in the past in different fields [8, 4, 5, 6].
- The computation of the value of each cell of the table T depends only on the three neighbour cells - $\mathcal{O}(1)$.
- For the DP matrix $T[0\,..\,m, 0\,..\,n]$, there are $m \times n$ values.
- The initialization phase requires time $\mathcal{O}(m + n)$.
- Hence, table T can be computed in $\mathcal{O}(m \times n)$ time and space.

Needleman-Wunsch algorithm & Global alignment

- Needleman and Wunsch simply re-formulated the edit distance problem [1, 3] in terms of maximizing similarity [4].
- Instead of minimizing distance we are **maximizing** similarity.
- Also known as **alignment score**.
- Sellers showed that the two problems are equivalent [6].
- The notion of distance is not suitable in biology.
- We rather utilize a notion of similarity between strings: dissimilarity is penalized and the similarity is favored; i.e. $\text{sub}(a, a) > 0$, $\text{sub}(a, b) < 0$, $\text{ins}(a) < 0$, $\text{del}(a) < 0$.
- The **Needleman-Wunsch** algorithm for **global alignment**.

Smith-Waterman algorithm & Local alignment

- Instead of considering a global alignment between x and y, in molecular biology it is often more relevant to determine a best alignment between a substring of x and a substring of y.
- **Local alignment** is more useful for dissimilar sequences that are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context.
- Similarly, the notion of distance between two strings is not suitable for biological applications.
- Similarly, we utilize a notion of similarity between strings for which the dissimilarity is penalized and similarity favored.
- The search for a similar substring consists then in maximizing the similarity (alignment score) between the strings.
- The **Smith-Waterman** algorithm for **local alignment** [10].

Smith-Waterman algorithm & Local alignment

Let x and y be two strings of lengths m and n, respectively.
We will assume a 1-based index only for these two strings:

$$x = x[1\,..\,m] \qquad\qquad y = y[1\,..\,n].$$

The cells of the DP matrix $S[0\,..\,m, 0\,..\,n]$ can be computed by:

$$S[i,j] = \begin{cases} 0 & : \quad i = j = 0 \\ 0 & : \quad 0 < i \leq m, j = 0 \\ 0 & : \quad 0 < j \leq n, i = 0 \\ \max \begin{cases} 0 \\ S[i-1, j-1] + \mathrm{sub}(x[i], y[j]) \\ S[i-1, j] + \mathrm{del}(x[i]) \\ S[i, j-1] + \mathrm{ins}(y[j]) \end{cases} & : \quad 0 < i \leq m, 0 < j \leq n \end{cases}$$

Smith-Waterman algorithm & Local alignment

Recall the formula for the global alignment!

$$
\begin{aligned}
T[0,0] &= 0, \\
T[i,0] &= T[i-1,0] + \mathrm{del}(x[i]), \\
T[0,j] &= T[0,j-1] + \mathrm{ins}(y[j]), \\
T[i,j] &= \min \begin{cases} T[i-1,j-1] + \mathrm{sub}(x[i], y[j]) \\ T[i-1,j] + \mathrm{del}(x[i]) \\ T[i,j-1] + \mathrm{ins}(y[j]). \end{cases}
\end{aligned}
$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

Smith-Waterman algorithm - Example

Let $x = \texttt{EAWACQGKL}$, $y = \texttt{ERDAWCQPGKWY}$, $\text{sub}(a, a) := 1$, $\text{sub}(a, b) := -3$, $\text{ins}(a) := \text{del}(a) := -1$, where Σ is the amino acids alphabet.

$$S[i][j] = \begin{cases} 0 & : \quad i = j = 0 \\ 0 & : \quad 0 < i \leq m, j = 0 \\ 0 & : \quad 0 < j \leq n, i = 0 \\ \max \begin{cases} 0 \\ S[i-1][j-1] + \text{sub}(x[i], y[j]) \\ S[i-1][j] + \text{del}(x[i]) \\ S[i][j-1] + \text{ins}(y[j]) \end{cases} & : \quad 0 < i \leq m, 0 < j \leq n \end{cases}$$

		0	1	2	3	4	5	6	7	8	9	10	11	12
	S		E	R	D	A	W	C	Q	P	G	K	W	Y
		0	0	0	0	0	0	0	0	0	0	0	0	0
0	E	0	1	0	0	0	0	0	0	0	0	0	0	0
1	A	0	0	0	0	1	0	0	0	0	0	0	0	0
2	W	0	0	0	0	0	2	1	0	0	0	0	1	0
3	A	0	0	0	0	1	1	0	0	0	0	0	0	0
4	C	0	0	0	0	0	0	2	1	0	0	0	0	0
5	Q	0	0	0	0	0	0	1	3	2	1	0	0	0
6	G	0	0	0	0	0	0	0	2	1	3	2	1	0
7	K	0	0	0	0	0	0	0	1	0	2	4	3	2
8	L	0	0	0	0	0	0	0	0	0	1	3	2	1

Smith-Waterman algorithm - Example (Traceback)

1. Locate one among the equally largest values in table S.
2. Traceback the path from the cell of this value by following the arrows, similarly as with the edit-distance algorithm.
3. Stop the scan on a zero value.

S	-	E	R	D	A	W	C	Q	P	G	K	W	Y
-	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	1	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	1	0	0	0	0	0	0	0	0
W	0	0	0	0	0	2	1	0	0	0	0	1	0
A	0	0	0	0	1	1	0	0	0	0	0	0	0
C	0	0	0	0	0	0	2	1	0	0	0	0	0
Q	0	0	0	0	0	0	1	3	2	1	0	0	0
G	0	0	0	0	0	0	0	2	1	3	2	1	0
K	0	0	0	0	0	0	0	1	0	2	4	3	2
L	0	0	0	0	0	0	0	0	0	1	3	2	1

$$\begin{pmatrix} \texttt{AWACQ-GK} \\ \texttt{AW-CQPGK} \end{pmatrix}$$

Smith-Waterman algorithm - Complexities

- The computation of the value of each cell of the table S depends only on the three neighbour cells - $\mathcal{O}(1)$.
- For the DP matrix $S[0\,..\,m, 0\,..\,n]$, there are $m \times n$ values.
- The initialization phase requires time $\mathcal{O}(m + n)$.
- Hence, table S can be computed in $\mathcal{O}(m \times n)$ time and space.

What happens when we want **all** local alignments of score $\geq t$?

- Repeat
  - Retrieve the highest scoring alignment
  - Set its trace to 0 and recalculate the affected cells.

The **Waterman-Eggert** algorithm for **local alignment** [9].

Contents

Banded dynamic programming

If no more than $d = 3$ *indels* (insertions/deletions) are allowed, only the following area of the matrix needs to be completed.

T	-	E	R	D	A	W	C	Q	P	G	K	W	Y
-	0	1	2	3									
E	1	0	1	2	3								
A	2	1	2	3	2	3							
W	3	2	3	4	3	2	3						
A		3	4	5	4	3	4	5					
C			5	6	5	4	3	4	5				
Q				7	6	5	4	3	4	5			
G					7	6	5	4	5	4	5		
K						7	6	5	6	5	4	5	
L							7	6	7	6	5	6	7

Size of $2d + 1$ band is $\mathcal{O}(d \times \min\{m, n\})$.

The running time is reduced from $\mathcal{O}(m \times n)$ to $\mathcal{O}(d \times \min\{m, n\})$.

This improvement was proposed by Ukkonen [7].

Alignment with gaps: Why?

- A *gap* is a sequence of consecutive insertions or deletions.
- The extensive use of alignments in biology has shown that it can be desirable to penalise the formation of long gaps
  - Rather than penalising individual insertions or deletions.
- A gap in a biological sequence is the absence (resp. presence) of a region, which is (resp. is not) present in another sequence.
- Gaps occur naturally in biology – diversity between individuals.
- A single mutational event can cause a gap
  - insertion or deletion of an entire region;
  - for instance, during the replication of DNA.

Alignment with gaps

- So far, we have penalised k contiguous spaces (1 gap) the same as k "dispersed" spaces.
- It makes sense to reduce the penalty for contiguous spaces.
- We can introduce a function

$$g : \mathbb{N} \to \mathbb{R}.$$

  whose value $g(k)$ indicates the cost of a gap of length k.
- The algorithm for edit distance is not suitable in this situation.

Alignment with gaps

Let x and y be two strings of lengths m and n, respectively.
We will assume a 1-based index only for these two strings:

$$x = x[1\,..\,m] \qquad\qquad y = y[1\,..\,n].$$

We utilise three matrices: D, I, and T:

- $D[i, j]$ indicates the cost of an optimal alignment between $x[0\,..\,i]$ and $y[0\,..\,j]$ ending with deletions of letters of x.
- $I[i, j]$ indicates the cost of an optimal alignment between $x[0\,..\,i]$ and $y[0\,..\,j]$ ending with insertions of letters of y.
- $T[i, j]$ indicates the cost of an optimal alignment between $x[0\,..\,i]$ and $y[0\,..\,j]$.

Alignment with gaps

The cells of the DP matrices $D[0\,..\,m][0\,..\,n]$, $I[0\,..\,m][0\,..\,n]$, and $T[0\,..\,m][0\,..\,n]$ can be computed by the following:

$$D[0,0] = D[i,0] = D[0,j] = I[0,0] = I[i,0] = I[0,j] = \infty,$$

and

$$T[0,0] = 0,$$

$$T[i,0] = g(i),$$

$$T[0,j] = g(j),$$

$$D[i,j] = \min\{\, T[\ell,j] + g(i-\ell) : \ell = 1,\ldots,i-1\},$$

$$I[i,j] = \min\{\, T[i,k] + g(j-k) : k = 1,\ldots,j-1\},$$

$$T[i,j] = \min\{\, T[i-1,j-1] + \mathsf{sub}(x[i],y[j]), D[i,j], I[i,j]\},$$

for $i = 1,\ldots,m$ and $j = 1,\ldots,n$.

Alignment with gaps

- If no restriction is done on the function g, we can check that the problem of the computation of an optimal alignment between x and y solves in time $\mathcal{O}(m \times n(m+n))$.
- Therefore, we will consider an *affine gap penalty* function $g(k)$. Affine function $g(k)$ for a gap of $k > 0$ positions is:

$$\textit{gap opening penalty} + (k-1) \times \textit{gap extension penalty}.$$

- *gap opening penalty*, denoted by f, is a positive constant representing the penalty for initiating the gap.
- *gap extension penalty*, denoted by h, is a positive constant representing the penalty proportional to the length of the gap.

Alignment with gaps

The gap penalty parameters remain fixed in aligning different positions. Therefore, the recurrence relations become:

$$D[0,0] = D[i,0] = D[0,j] = I[0,0] = I[i,0] = I[0,j] = \infty$$

$$D[i,j] = \min\{D[i-1,j] + h,\, T[i-1,j] + f\},$$

$$I[i,j] = \min\{I[i,j-1] + h,\, T[i,j-1] + f\},$$

$$T[i,j] = \min\{T[i-1,j-1] + \mathrm{sub}(x[i], y[j]), D[i,j], I[i,j]\},$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$, and

$$T[0,0] = 0,$$

$$T[1,0] = T[0,1] = f,$$

$$T[i,0] = T[i-1,0] + h,$$

$$T[0,j] = T[0,j-1] + h,$$

for $i = 2, \ldots, m$ and $j = 2, \ldots, n$.

Alignment with gaps - Example

Let $x = \texttt{EAWACQGKL}$, $y = \texttt{ERDAWCQPGKWY}$, $\text{sub}(a, b) := 3$, $f := 3$, and $h := 1$, where Σ is the amino acids alphabet.

$$D[0,0] = D[i,0] = D[0,j] = \infty,$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

		0	1	2	3	4	5	6	7	8	9	10	11	12
	D		E	R	D	A	W	C	Q	P	G	K	W	Y
0		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	E	∞												
2	A	∞												
3	W	∞												
4	A	∞												
5	C	∞												
6	Q	∞												
7	G	∞												
8	K	∞												
9	L	∞												

Alignment with gaps - Example

Let $x = \texttt{EAWACQGKL}$, $y = \texttt{ERDAWCQPGKWY}$, $\text{sub}(a, b) := 3$, $f := 3$, and $h := 1$, where Σ is the amino acids alphabet.

$$I[0, 0] = I[i, 0] = I[0, j] = \infty,$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

		0	1	2	3	4	5	6	7	8	9	10	11	12
	I		E	R	D	A	W	C	Q	P	G	K	W	Y
0		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	E	∞												
2	A	∞												
3	W	∞												
4	A	∞												
5	C	∞												
6	Q	∞												
7	G	∞												
8	K	∞												
9	L	∞												

Alignment with gaps - Example

Let $x = \texttt{EAWACQGKL}$, $y = \texttt{ERDAWCQPGKWY}$, $\mathrm{sub}(a, b) := 3$, $f := 3$, and $h := 1$, where Σ is the amino acids alphabet.

$$T[0,0] = 0,$$

$$T[1,0] = T[0,1] = f,$$

$$T[i,0] = T[i-1,0] + h,$$

$$T[0,j] = T[0,j-1] + h,$$

for $i = 2, \ldots, m$ and $j = 2, \ldots, n$.

		0	1	2	3	4	5	6	7	8	9	10	11	12
	T		E	R	D	A	W	C	Q	P	G	K	W	Y
0		0	3	4	5	6	7	8	9	10	11	12	13	14
1	E	3												
2	A	4												
3	W	5												
4	A	6												
5	C	7												
6	Q	8												
7	G	9												
8	K	10												
9	L	11												

Alignment with gaps

- The general penalty function can possibly have a different penalty for each additional gap.
  - The algorithm has to check for all possible gap lengths!
- The problem with function g as an affine function can be solved in time and space $\mathcal{O}(m \times n)$.
- This function penalises the opening of a gap by f and to penalise differently the extension of a gap by h.
- In real applications, we choose $h < f$.
- This improvement for aligning biological sequences via using the affine function was proposed by Gotoh in [2].

Contents

Overview

- Pairwise sequence alignment is the process of comparing two strings of letters to infer or visualize their similarity.
- There exist two main distances for comparing two strings — the edit distance and the Hamming distance.
- A different formulation of the edit distance is to maximize the similarity of the two strings — global alignment (Needleman-Wunsch algorithm) — instead of minimizing the distance between the two strings.
- Instead of considering a global alignment between two strings, in biological applications it is often more relevant to determine a best alignment between substrings of the two strings — local alignment (Smith-Waterman algorithm).

Overview

- An improvement can be made if we can restrict the number of insertions and deletions: banded dynamic programming.
- It is desirable to penalise the formation of long gaps rather than penalising individual insertions or deletions of letters.
- If no restriction is done on the gap penalty function we get an inefficient algorithm.
- An improvement can be made if we make use of an affine gap penalty function instead.

Bibliography I


F. J. Damerau.
A technique for computer detection and correction of spelling errors.
Commun. ACM, 7(3):171–176, 1964.


O. Gotoh.
An improved algorithm for matching biological sequences.
Journal of molecular biology, 162(3):705–708, 1982.


V. I. Levenshtein.
Binary codes capable of correcting deletions, insertions, and reversals.
Technical Report 8, Soviet Physics Doklady, 1966.

Bibliography II


S. B. Needleman and C. D. Wunsch.
A general method applicable to the search for similarities in the amino acid sequence of two proteins.
Journal of Molecular Biology, 48(3):443–453, 1970.


D. Sankoff.
Matching Sequences under Deletion/Insertion Constraints.
Proceedings of the National Academy of Sciences of the United States of America, 69(1):4–6, 1972.


P. H. Sellers.
On the theory and computation of evolutionary distances.
SIAM Journal on Applied Mathematics, 26(4):787–793, 1974.

Bibliography III


E. Ukkonen.
On approximate string matching.
In M. Karpinski, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 487–495. Springer Berlin Heidelberg, 1983.


T. Vintsyuk.
Speech discrimination by dynamic programming.
Cybernetics, 4:52–57, 1968.


M. S. Waterman and M. Eggert.
A new algorithm for best subsequence alignments with application to trna-rrna comparisons.
Journal of Molecular Biology, 197(4):723–728, 1987.

Bibliography IV


M. S. Waterman and T. F. Smith.
Identification of common molecular subsequences.
Journal of Molecular Biology, 147(1):195–197, 1981.