Combinatorial pattern matching is the search for exact or approximate occurrences of a given pattern within a given text.

When it comes to biological sequences, both the pattern and the text are sequences and the pattern matching problem becomes one of finding the occurrences of a sequence within another sequence.

For instance, scanning a protein sequence for the presence of a known pattern can help annotate both the protein and the corresponding genome, and finding a sequence within another sequence can help in assessing their similarities and differences.

Definition

The similarities and differences between two sequences can be assessed by computing a distance measure between the two sequences.

The edit distance is based on the elementary edit operations of inserting an element in a sequence, deleting an element from a sequence, and substituting another element for an element in a sequence.

The alignment of two sequences is an explicit description of the correspondence between the elements of the two sequences, together with the positions at which element insertions and deletions take place in each sequence.

Definition

The edit distance is based on the insertion, deletion, and substitution of elements in the two sequences under comparison.

The number and type of edit operations needed to transform one sequence into the other reveal similarities and differences between two sequences.

There are several types of edit distance between sequences, depending on the type of edit operations allowed.

The Hamming distance between two sequences of the same length is defined as the number of positions at which the two sequences differ.

This is the same as the smallest number of element substitutions needed to transform one sequence into the other, meaning that insertions and deletions are forbidden and only substitutions are allowed in the Hamming distance.

Example

At least 12 element substitutions are needed to transform the DNA sequence TGCTTCTGACTATAATAG into GCTTCCGGCTCGTATAAT, as shown in the following alignment. Therefore, the Hamming distance between the two sequences is 12.

```
TGCTTCTGACTATAATAG
||| | | ||||  || |
GCTTCCGGCTCGTATAAT
```

Algorithm

The Hamming distance between two sequences of the same length can be computed by traversing the sequences and counting the number of positions at which they differ.

The Hamming distance d between two sequences S_1 and S_2 is set to -1 when they have different lengths; otherwise, it is obtained as the number of sequence positions i such that $S_1[i] \neq S_2[i]$.

Algorithm

```
**function** hamming_distance($S_1, S_2$)
    $n \leftarrow length(S_1)$
    **if** $n \neq length(S_2)$ **then**
        $d \leftarrow -1$
    **else**
        $d \leftarrow 0$
        **for** $i \leftarrow 1$ **to** $n$ **do**
            **if** $S_1[i] \neq S_2[i]$ **then**
                $d \leftarrow d + 1$
    **return** $d$
```

Definition

The Levenshtein distance between two sequences (not necessarily of the same length) is defined as the smallest number of element insertions and deletions needed to transform one sequence into the other.

Unlike the Hamming distance, in which only element substitutions are allowed, in the Levenshtein distance only element insertions and deletions are allowed.

Example

The DNA sequence GCTTCCGGCTCGTATAATGTGTGG can be transformed into TGCTTCTGACTATAATAG by 4 insertions and 10 deletions, as shown in the following alignment, and this is the least possible number of insertions and deletions to transform one of these sequences into the other. Therefore, the Levenshtein distance between them is 14.

```
-GCTTCC-GG-CTCGTATAAT-GTGTGG
 |||||  |  ||   ||||| |
TGCTTC-TG-ACT---ATAATAG-----
```

Algorithm

Given two sequences S_1 and S_2, assume a prefix $S_1[1,\ldots,i-1]$ can be transformed into a prefix $S_2[1,\ldots,j]$ by x insertions and deletions, a prefix $S_1[1,\ldots,i]$ can be transformed into a prefix $S_2[1,\ldots,j-1]$ by y insertions and deletions, and $S_1[1,\ldots,i-1]$ can be transformed into $S_2[1,\ldots,j-1]$ by z insertions and deletions.

Then $S_1[1,\ldots,i]$ can also be transformed into $S_2[1,\ldots,j]$ by z insertions and deletions if $S_1[i]=S_2[j]$ or else by the x edit operations plus the insertion of element $S_1[i]$ or the y edit operations plus the deletion of element $S_2[j]$.

Algorithm

In general, the Levenshtein distance d between two sequences S_1 and S_2 is given by the recurrence

$$
\begin{aligned}
& d(S_1[1,\ldots,i], S_2[1,\ldots,j]) = \\
= \quad & \min \begin{cases} d(S_1[1,\ldots,i-1], S_2[1,\ldots,j]) + 1, \\ d(S_1[1,\ldots,i], S_2[1,\ldots,j-1]) + 1, \\ d(S_1[1,\ldots,i-1], S_2[1,\ldots,j-1]) & \text{if } S_1[i] = S_2[j] \end{cases}
\end{aligned}
$$

where $d(S_1[1,\ldots,i], S_2[1,\ldots,j]) = 0$ if both $i = 0$ and $j = 0$, $d(S_1[1,\ldots,i], S_2[1,\ldots,j]) = i$ if $i \neq 0$ and $j = 0$, and $d(S_1[1,\ldots,i], S_2[1,\ldots,j]) = j$ if $i = 0$ and $j \neq 0$.

Computation of this recurrence by dynamic programming involves the use of a dynamic programming table to store each $d(S_1[1,\ldots,i], S_2[1,\ldots,j])$, for $1 \leqslant i \leqslant n_1$ and $1 \leqslant j \leqslant n_2$, where n_1 is the length of S_1 and n_2 is the length of S_2.

Example

The Levenshtein distance between the DNA sequences GCTTCCGGCTCGTATAATGTGTGG and TGCTTCTGACTATAATAG is 14, as shown in entry $(24, 18)$ of the following dynamic programming table.

Example

			T	G	C	T	T	C	T	G	A	C	T	A	T	A	A	T	A	G
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
G	1	1	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	2	2	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	3	3	2	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	4	4	3	4	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	5	5	4	5	4	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13
C	6	6	5	6	5	4	3	2	3	4	5	4	5	6	7	8	9	10	11	12
G	7	7	6	5	6	5	4	3	4	3	4	5	6	7	8	9	10	11	12	11
G	8	8	7	6	7	6	5	4	5	4	5	6	7	8	9	10	11	12	13	12
C	9	9	8	7	6	7	6	5	6	5	6	5	6	7	8	9	10	11	12	13
T	10	10	9	8	7	6	7	6	5	6	7	6	5	6	7	8	9	10	11	12
C	11	11	10	9	8	7	8	7	6	7	8	7	6	7	8	9	10	11	12	13
G	12	12	11	10	9	8	9	8	7	6	7	8	7	8	9	10	11	12	13	12
T	13	13	12	11	10	9	8	9	8	7	8	9	8	9	8	9	10	11	12	13
A	14	14	13	12	11	10	9	10	9	8	7	8	9	8	9	8	9	10	11	12
T	15	15	14	13	12	11	10	11	10	9	8	9	8	9	8	9	10	9	10	11
A	16	16	15	14	13	12	11	12	11	10	9	10	9	8	9	8	9	10	9	10
A	17	17	16	15	14	13	12	13	12	11	10	11	10	9	10	9	8	9	10	11
T	18	18	17	16	15	14	13	14	13	12	11	12	11	10	9	10	9	8	9	10
G	19	19	18	17	16	15	14	15	14	13	12	13	12	11	10	11	10	9	10	9
T	20	20	19	18	17	16	15	16	15	14	13	14	13	12	11	12	11	10	11	10
G	21	21	20	19	18	17	16	17	16	15	14	15	14	13	12	13	12	11	12	11
T	22	22	21	20	19	18	17	18	17	16	15	16	15	14	13	14	13	12	13	12
G	23	23	22	21	20	19	18	19	18	17	16	17	16	15	14	15	14	13	14	13
G	24	24	23	22	21	20	19	20	19	18	17	18	17	16	15	16	15	14	15	14

Algorithm

The dynamic programming table D is filled in for each $0 \leqslant i \leqslant n_1$ and $0 \leqslant j \leqslant n_2$, and the Levenshtein distance between S_1 and S_2 is stored in entry $D[n_1, n_2]$.

Algorithm

```
function levenshtein_distance($S_1, S_2$)
    $n_1 \leftarrow$ length($S_1$)
    $n_2 \leftarrow$ length($S_2$)
    $D[0,0] \leftarrow 0$
    for $i \leftarrow 1$ to $n_1$ do
        $D[i,0] \leftarrow i$
    for $j \leftarrow 1$ to $n_2$ do
        $D[0,j] \leftarrow j$
    for $i \leftarrow 1$ to $n_1$ do
        for $j \leftarrow 1$ to $n_2$ do
            if $S_1[i] = S_2[j]$ then
                $D[i,j] \leftarrow D[i-1,j-1]$
            else
                $D[i,j] \leftarrow \min(D[i-1,j]+1, D[i,j-1]+1)$
    return $D[n_1,n_2]$
```

Definition

In general, the edit distance between two sequences (not necessarily of the same length) is defined as the smallest number of insertions, deletions, and substitutions needed to transform one sequence into the other.

The edit distance thus combines the Hamming distance, in which only element substitutions are allowed, with the Levenshtein distance, in which only element insertions and deletions are allowed.

Example

The DNA sequence GCTTCCGGCTCGTATAATGTGTGG can be transformed into TGCTTCTGACTATAATAG by 1 insertion, 7 deletions, and 3 substitutions, as shown in the following alignment, and this is the least possible number of insertions, deletions, and substitutions to transform one of these sequences into the other. Therefore, the edit distance between them is 11.

```
-GCTTCCGGCTCGTATAATGTGTGG
 |||||*|*||   |||||* |
TGCTTCTGACT---ATAATA-G---
```

Algorithm

Given two sequences S_1 and S_2, assume a prefix $S_1[1,\dots,i-1]$ can be transformed into a prefix $S_2[1,\dots,j]$ by x insertions, deletions, and substitutions, a prefix $S_1[1,\dots,i]$ can be transformed into a prefix $S_2[1,\dots,j-1]$ by y insertions, deletions, and substitutions, and $S_1[1,\dots,i-1]$ can be transformed into $S_2[1,\dots,j-1]$ by z insertions, deletions, and substitutions.

Then $S_1[1,\dots,i]$ can also be transformed into $S_2[1,\dots,j]$ by z edit operations if $S_1[i] = S_2[j]$ or else by the x edit operations plus the insertion of element $S_1[i]$, the y edit operations plus the deletion of element $S_2[j]$, or the z edit operations plus the substitution of element $S_2[j]$ for element $S_1[i]$.

Algorithm

In general, the edit distance d between two sequences S_1 and S_2 is given by the recurrence

$$
\begin{aligned}
& d(S_1[1,\ldots,i],S_2[1,\ldots,j]) = \\
&= \quad \min \begin{cases} d(S_1[1,\ldots,i-1],S_2[1,\ldots,j])+1, & \\ d(S_1[1,\ldots,i],S_2[1,\ldots,j-1])+1, & \\ d(S_1[1,\ldots,i-1],S_2[1,\ldots,j-1]) & \text{if } S_1[i]=S_2[j] \\ d(S_1[1,\ldots,i-1],S_2[1,\ldots,j-1])+1 & \text{if } S_1[i]\neq S_2[j] \end{cases}
\end{aligned}
$$

where $d(S_1[1,\ldots,i],S_2[1,\ldots,j])=0$ if both $i=0$ and $j=0$, $d(S_1[1,\ldots,i], S_2[1,\ldots,j])=i$ if $i\neq 0$ and $j=0$, and $d(S_1[1,\ldots,i],S_2[1,\ldots,j])=j$ if $i=0$ and $j\neq 0$.

Computation of this recurrence by dynamic programming involves the use of a dynamic programming table to store each $d(S_1[1,\ldots,i],S_2[1,\ldots,j])$, for $1\leqslant i\leqslant n_1$ and $1\leqslant j\leqslant n_2$, where n_1 is the length of S_1 and n_2 is the length of S_2.

Example

The edit distance between the DNA sequences GCTTCCGGCTCGTATAATGTGTGG and TGCTTCTGACTATAATAG is 11, as shown in entry $(24, 18)$ of the following dynamic programming table.

Example

			T	G	C	T	T	C	T	G	A	C	T	A	T	A	A	T	A	G
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
G	1	1	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	2	2	2	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	3	3	2	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	4	4	3	3	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	5	5	4	4	3	3	2	1	2	3	4	5	6	7	8	9	10	11	12	13
C	6	6	5	5	4	4	3	2	2	3	4	4	5	6	7	8	9	10	11	12
G	7	7	6	5	5	5	4	3	3	2	3	4	5	6	7	8	9	10	11	11
G	8	8	7	6	6	6	5	4	4	3	3	4	5	6	7	8	9	10	11	11
C	9	9	8	7	6	7	6	5	5	4	4	3	4	5	6	7	8	9	10	11
T	10	10	9	8	7	6	7	6	5	5	5	4	3	4	5	6	7	8	9	10
C	11	11	10	9	8	7	7	7	6	6	6	5	4	4	5	6	7	8	9	10
G	12	12	11	10	9	8	8	8	7	6	7	6	5	5	5	6	7	8	9	9
T	13	13	12	11	10	9	8	9	8	7	7	7	6	6	5	6	7	7	8	9
A	14	14	13	12	11	10	9	9	9	8	7	8	7	6	6	5	6	7	7	8
T	15	15	14	13	12	11	10	10	9	9	8	8	8	7	6	6	6	6	7	8
A	16	16	15	14	13	12	11	11	10	10	9	9	9	8	7	6	6	7	6	7
A	17	17	16	15	14	13	12	12	11	11	10	10	10	9	8	7	6	7	7	7
T	18	18	17	16	15	14	13	13	12	12	11	11	10	10	9	8	7	6	7	8
G	19	19	18	17	16	15	14	14	13	12	12	12	11	11	10	9	8	7	7	7
T	20	20	19	18	17	16	15	15	14	13	13	13	12	12	11	10	9	8	8	8
G	21	21	20	19	18	17	16	16	15	14	14	14	13	13	12	11	10	9	9	8
T	22	22	21	20	19	18	17	17	16	15	15	15	14	14	13	12	11	10	10	9
G	23	23	22	21	20	19	18	18	17	16	16	16	15	15	14	13	12	11	11	10
G	24	24	23	22	21	20	19	19	18	17	17	17	16	16	15	14	13	12	12	11

Algorithm

The dynamic programming table D is filled in for each $0 \leqslant i \leqslant n_1$ and $0 \leqslant j \leqslant n_2$, and the edit distance between S_1 and S_2 is stored in entry $D[n_1, n_2]$.

Algorithm

```
function edit_distance($S_1, S_2$)
    $n_1 \leftarrow$ length($S_1$); $n_2 \leftarrow$ length($S_2$)
    $D[0,0] \leftarrow 0$
    for $i \leftarrow 1$ to $n_1$ do
        $D[i,0] \leftarrow i$
    for $j \leftarrow 1$ to $n_2$ do
        $D[0,j] \leftarrow j$
    for $i \leftarrow 1$ to $n_1$ do
        for $j \leftarrow 1$ to $n_2$ do
            $D[i,j] \leftarrow \min(D[i-1,j]+1, D[i,j-1]+1)$
            if $S_1[i] = S_2[j]$ then
                $D[i,j] \leftarrow \min(D[i,j], D[i-1,j-1])$
            else
                $D[i,j] \leftarrow \min(D[i,j], D[i-1,j-1]+1)$
    return $D[n_1, n_2]$
```

Definition

An alignment of two sequences is an arrangement of the two sequences as rows of a matrix, with additional gaps (dashes) between the elements to make some or all of the remaining (aligned) columns contain identical elements but with no column gapped in both sequences.

A dash in the first sequence of an alignment corresponds to the insertion of the opposite element into the first sequence, a dash in the second sequence of an alignment corresponds to the deletion of the opposite element from the second sequence, and two mismatched elements opposite in an alignment correspond to a substitution of the element in the second sequence for the element in the first sequence.

Definition

The Levenshtein distance between two sequences is thus given by an alignment of the two sequences with the smallest possible number of dashes (insertions or deletions) and with no mismatched elements (substitutions), while the edit distance between two sequences is given by an alignment of the two sequences with the smallest possible number of dashes (insertions or deletions) plus mismatched elements (substitutions).

Example

The DNA sequence GCTTCCGGCTCGTATAATGTGTGG can be transformed into TGCTTCTGACTATAATAG by inserting T, T, A, A before (original) positions 1, 7, 9, 19, and deleting C, G, C, G, T, T, G, T, G, G at (original) positions 6, 8, 11, 12, 13, 20, 21, 22, 23, 24, as shown in the following alignment.

```
-GCTTCC-GG-CTCGTATAAT-GTGTGG
 |||||  |  ||    ||||| |
TGCTTC-TG-ACT---ATAATAG-----
```

Sequence GCTTCCGGCTCGTATAATGTGTGG can also be transformed into TGCTTCTGACTATAATAG by inserting T before position 1; substituting T for C at position 6; substituting A for G at position 8; deleting C, G, T at positions 11, 12, 13; substituting A for G at position 19; deleting T at position 20; and deleting T, G, G at positions 22, 23, 24, as shown in the following alignment.

```
-GCTTCCGGCTCGTATAATGTGTGG
 |||||*|*||   |||||* |
TGCTTCTGACT---ATAATA-G---
```

Algorithm

An alignment of two sequences can be obtained from the dynamic programming table, already filled in when computing the Levenshtein distance or the edit distance between the two sequences, by tracing the sequence of edit operations from the final (bottom right) position back to the initial (top left) position. In the trace back at position i of S_1 and position j of S_2,

- $D(S_1[1,\ldots,i], S_2[1,\ldots,j]) = D(S_1[1,\ldots,i], S_2[1,\ldots,j-1]) + 1$ indicates the insertion of a dash into S_1, and
- $D(S_1[1,\ldots,i], S_2[1,\ldots,j]) = D(S_1[1,\ldots,i-1], S_2[1,\ldots,j]) + 1$ indicates the insertion of a dash into S_2.

Since there is a choice of moving left or moving up (if the previous conditions are fulfilled) and also moving in diagonal, if either $S_1[i] = S_2[j]$ and
$D(S_1[1,\ldots,i], S_2[1,\ldots,j]) = D(S_1[1,\ldots,i-1], S_2[1,\ldots,j-1])$ or
$S_1[i] \neq S_2[j]$ and
$D(S_1[1,\ldots,i], S_2[1,\ldots,j]) = D(S_1[1,\ldots,i-1], S_2[1,\ldots,j-1]) + 1$, several alignments may be implicit in a single dynamic programming table.

Example

The Levenshtein distance between the DNA sequences GCTTCCGGCTCGTATAATGTGTGG and TGCTTCTGACTATAATAG gives 1,430 different alignments.

Each such alignment can be obtained by following a different path of shaded entries from the final (bottom right) back to the initial (top left) entry of the following dynamic programming table, inserting a dash into S_1 when moving to the left and inserting a dash into S_2 when moving up.

For instance, by moving to the left if possible, otherwise up if possible, or else in diagonal, the following alignment of the two sequences is obtained, where matches are indicated with a vertical bar.

```
-GCTTCC-GG-CTCGTATAAT-GTGTGG
 |||||  |  ||   ||||| |
TGCTTC-TG-ACT---ATAATAG-----
```

Example

			T	G	C	T	T	C	T	G	A	C	T	A	T	A	A	T	A	G
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	0	**0**	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
G	1	1	2	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	2	2	3	2	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	3	3	2	3	2	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	4	4	3	4	3	2	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14
C	5	5	4	5	4	3	**2**	**1**	**2**	3	4	5	6	7	8	9	10	11	12	13
C	6	6	5	6	5	4	**3**	**2**	**3**	4	5	4	5	6	7	8	9	10	11	12
G	7	7	6	5	6	5	**4**	**3**	**4**	**3**	**4**	5	6	7	8	9	10	11	12	11
G	8	8	7	6	7	6	**5**	**4**	5	**4**	**5**	6	7	8	9	10	11	12	13	12
C	9	9	8	7	6	7	6	**5**	6	**5**	**6**	**5**	6	7	8	9	10	11	12	13
T	10	10	9	8	7	6	7	6	**5**	**6**	**7**	**6**	**5**	6	7	8	9	10	11	12
C	11	11	10	9	8	7	8	7	**6**	7	8	**7**	**6**	7	8	9	10	11	12	13
G	12	12	11	10	9	8	9	8	7	**6**	**7**	**8**	**7**	8	9	10	11	12	13	12
T	13	13	12	11	10	9	8	9	8	7	8	9	**8**	9	8	9	10	11	12	13
A	14	14	13	12	11	10	9	10	9	8	7	8	9	**8**	9	8	9	10	11	12
T	15	15	14	13	12	11	10	11	10	9	8	9	8	9	**8**	9	10	9	10	11
A	16	16	15	14	13	12	11	12	11	10	9	10	9	8	9	**8**	9	10	9	10
A	17	17	16	15	14	13	12	13	12	11	10	11	10	9	10	9	**8**	9	10	11
T	18	18	17	16	15	14	13	14	13	12	11	12	11	10	9	10	**9**	**8**	**9**	10
G	19	19	18	17	16	15	14	15	14	13	12	13	12	11	10	11	**10**	**9**	**10**	**9**
T	20	20	19	18	17	16	15	16	15	14	13	14	13	12	11	12	**11**	**10**	**11**	**10**
G	21	21	20	19	18	17	16	17	16	15	14	15	14	13	12	13	**12**	**11**	**12**	**11**
T	22	22	21	20	19	18	17	18	17	16	15	16	15	14	13	14	13	**12**	**13**	**12**
G	23	23	22	21	20	19	18	19	18	17	16	17	16	15	14	15	14	**13**	**14**	**13**
G	24	24	23	22	21	20	19	20	19	18	17	18	17	16	15	16	15	14	15	**14**

Example

The edit distance between the DNA sequences GCTTCCGGCTCGTATAATGTGTGG and TGCTTCTGACTATAATAG gives 187 different alignments.

Each such alignment can be obtained by following a different path of shaded entries from the final (bottom right) back to the initial (top left) entry of the following dynamic programming table, inserting a dash into S_1 when moving to the left and inserting a dash into S_2 when moving up.

For instance, by moving to the left if possible, otherwise up if possible, or else in diagonal, the following alignment of the two sequences is obtained, where matches are indicated with a vertical bar and mismatches with an asterisk.

```
-GCTTCCGGCTCGTATAATGTGTGG
 |||||*|*||   |||||* |
TGCTTCTGACT---ATAATA-G---
```

Example

			T	G	C	T	T	C	T	G	A	C	T	A	T	A	A	T	A	G
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	0	**0**	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
G	1	1	1	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	2	2	2	2	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	3	3	2	3	2	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	4	4	3	3	3	2	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14
C	5	5	4	4	3	3	**2**	**1**	2	3	4	5	6	7	8	9	10	11	12	13
C	6	6	5	5	4	4	3	**2**	**2**	3	4	4	5	6	7	8	9	10	11	12
G	7	7	6	5	5	5	4	3	**3**	**2**	3	4	5	6	7	8	9	10	11	11
G	8	8	7	6	6	6	5	4	4	**3**	**3**	4	5	6	7	8	9	10	11	11
C	9	9	8	7	6	7	6	5	5	**4**	**4**	**3**	4	5	6	7	8	9	10	11
T	10	10	9	8	7	6	7	6	5	5	**5**	**4**	**3**	4	5	6	7	8	9	10
C	11	11	10	9	8	7	7	7	6	6	6	**5**	**4**	4	5	6	7	8	9	10
G	12	12	11	10	9	8	8	8	7	6	7	**6**	**5**	5	5	6	7	8	9	9
T	13	13	12	11	10	9	8	9	8	7	7	7	**6**	6	5	6	7	7	8	9
A	14	14	13	12	11	10	9	9	9	8	7	8	7	**6**	6	5	6	7	7	8
T	15	15	14	13	12	11	10	10	9	9	8	8	8	7	**6**	6	6	6	7	8
A	16	16	15	14	13	12	11	11	10	10	9	9	9	8	7	**6**	6	7	6	7
A	17	17	16	15	14	13	12	12	11	11	10	10	10	9	8	7	**6**	7	7	7
T	18	18	17	16	15	14	13	13	12	12	11	11	10	10	9	8	**7**	**6**	7	8
G	19	19	18	17	16	15	14	14	13	12	12	12	11	11	10	9	**8**	**7**	**7**	7
T	20	20	19	18	17	16	15	15	14	13	13	13	12	12	11	10	**9**	**8**	**8**	8
G	21	21	20	19	18	17	16	16	15	14	14	14	13	13	12	11	**10**	**9**	**9**	**8**
T	22	22	21	20	19	18	17	17	16	15	15	15	14	14	13	12	11	**10**	**10**	**9**
G	23	23	22	21	20	19	18	18	17	16	16	16	15	15	14	13	12	11	**11**	**10**
G	24	24	23	22	21	20	19	19	18	17	17	17	16	16	15	14	13	12	12	**11**

Algorithm

Once the Levenshtein distance or the edit distance between two sequences has been computed, an alignment of the two sequences can be obtained by tracing back the dynamic programming table from the final (bottom right) to the initial (top left) entry, inserting a dash into S_1 and moving up if $D(S_1[1,\ldots,i],S_2[1,\ldots,j]) = D(S_1[1,\ldots,i],S_2[1,\ldots,j-1])+1$, otherwise inserting a dash into S_2 and moving to the left if $D(S_1[1,\ldots,i],S_2[1,\ldots,j]) = D(S_1[1,\ldots,i-1],S_2[1,\ldots,j])+1$, or else moving up and to the left, in diagonal.

Once the first row or the first column has been reached, further movements up (after reaching the first column) or to the left (after reaching the first row) may be needed in order to finish computing the alignment.

Recall that the dynamic programming table D of a sequence S_1 of length n_1 and a sequence S_2 of length n_2 has n_1+1 rows numbered $0,\ldots,n_1$ and n_2+1 columns numbered $0,\ldots,n_2$.

Algorithm

```
**function** alignment($S_1, S_2, D$)
    $i \leftarrow$ length($S_1$)
    $j \leftarrow$ length($S_2$)
    $T_1 \leftarrow T_2 \leftarrow$ “”
    **while** $i > 1$ **and** $j > 1$ **do**
        **if** $D[i,j] = D[i,j-1]+1$ **then**
            $T_1 \leftarrow$ “–” $\cdot T_1$
            $T_2 \leftarrow S_2[j-1] \cdot T_2$
            $j \leftarrow j-1$
        **else if** $D[i,j] = D[i-1,j]+1$ **then**
            $T_1 \leftarrow S_1[i-1] \cdot T_1$
            $T_2 \leftarrow$ “–” $\cdot T_2$
            $i \leftarrow i-1$
```

Algorithm

```
        else
            $T_1 \leftarrow S_1[i-1] \cdot T_1$
            $T_2 \leftarrow S_2[j-1] \cdot T_2$
            $i \leftarrow i-1$
            $j \leftarrow j-1$
    while $j > 1$ do
        $T_1 \leftarrow$ "–" $\cdot T_1$
        $T_2 \leftarrow S_2[j-1] \cdot T_2$
        $j \leftarrow j-1$
    while $i > 1$ do
        $T_1 \leftarrow S_1[i-1] \cdot T_1$
        $T_2 \leftarrow$ "–" $\cdot T_2$
        $i \leftarrow i-1$
    return $(T_1, T_2)$
```

Definition

The assessment of similarities and differences between two sequences based on the computation of an edit distance or an alignment of the two sequences can also reflect the relative frequencies with which nucleotide substitutions (for DNA and RNA sequences) or amino acid substitutions (for protein sequences) take place.

This can be achieved by assigning a weight or score to each edit operation, depending on either the type of edit operation (element insertion, deletion, substitution) or the actual elements (nucleotides, amino acids) involved in the edit operation.

These generalized forms of edit distance and alignment can be computed by a straightforward extension to the edit distance recurrences and corresponding algorithms, where the particular score or weight of the edit operation upon the actual elements is substituted for the summand value 1.

Definition

Both the Levenshtein distance and the edit distance between two sequences give a global alignment of the sequences, that is, an alignment in which the overall number or the total score or weight of the insertions, deletions, and mismatches is as small as possible.

In a local alignment, on the other hand, only some subsequences of the two sequences are aligned: those subsequences that give the smallest possible edit distance.

Two sequences might actually have a large (global) edit distance but still contain subsequences at small (local) edit distance.

Definition

In the formulation of local alignment in terms of edit distance, however, a local alignment over short subsequences cannot always be distinguished from a local alignment over longer subsequences.

For instance, the edit distance between two sequences that contain identical subsequences might be the same, no matter the length of the common subsequences, while the sequences are more similar to each other the longer the common subsequences.

The shift from distances to similarities, where matches have a positive weight and insertions, deletions, and mismatches have a negative score, overcomes this problem. Insertions and deletions are also called gaps, because they introduce a gap (usually represented as a dash) in a sequence alignment.

Definition

In general, the local alignment of two sequences defines stretches of high similarity between the sequences, where a certain subsequence of the first sequence is aligned to a subsequence of the second sequence with a high combined weight or score of matches, mismatches, and gaps.

Example

Prefix GCTTCCGGCTCGTATAAT of DNA sequence GCTTCCGGCTCGTATAATGTGTGG can be aligned to subsequence GCTTCTGACTATAAT of DNA sequence TGCTTCTGACTATAATAG with 13 matches and only 2 mismatches and 3 gaps, as shown in the following local alignment.

```
GCTTCCGGCTCGTATAAT
||||  |*| *| ||||||
GCTT-CTG-AC-TATAAT
```

Algorithm

Given two sequences S_1 and S_2, assume that the largest possible score when aligning a suffix of prefix $S_1[1,\ldots,i-1]$ to a suffix of prefix $S_2[1,\ldots,j]$ is x, the largest possible score when aligning a suffix of prefix $S_1[1,\ldots,i]$ to a suffix of prefix $S_2[1,\ldots,j-1]$ is y, and the largest possible score when aligning a suffix of $S_1[1,\ldots,i-1]$ to a suffix of $S_2[1,\ldots,j-1]$ is z.

Then the largest possible score for aligning a suffix of $S_1[1,\ldots,i]$ to a suffix of $S_2[1,\ldots,j]$ is the largest value among z plus either the match weight, if $S_1[i] = S_2[j]$, or the mismatch weight, if $S_1[i] \neq S_2[j]$; y plus the gap score, for deleting element $S_1[i]$ from S_2; x plus the gap score, for inserting element $S_2[j]$ into S_1; and zero, to account for any negative values.

Algorithm

In general, the suffix similarity s between two sequences S_1 and S_2 is given by the recurrence

$$s(S_1[1,\ldots,i],S_2[1,\ldots,j]) =$$

$$= \quad \max \begin{cases} s(S_1[1,\ldots,i-1],S_2[1,\ldots,j]) + \mathit{gap}, & \\ s(S_1[1,\ldots,i],S_2[1,\ldots,j-1]) + \mathit{gap}, & \\ s(S_1[1,\ldots,i-1],S_2[1,\ldots,j-1]) + \mathit{match}, & \text{if } S_1[i] = S_2[j] \\ s(S_1[1,\ldots,i-1],S_2[1,\ldots,j-1]) + \mathit{mismatch}, & \text{if } S_1[i] \neq S_2[j] \\ 0 & \end{cases}$$

where match is the positive match score, mismatch is the negative mismatch score, gap is the negative gap score, and $s(S_1[1,\ldots,i],S_2[1,\ldots,j]) = 0$ if $i = 0$ or $j = 0$.

Algorithm

Computation of this recurrence by dynamic programming involves the use of a dynamic programming table to store each $s(S_1[1,\ldots,i], S_2[1,\ldots,j])$, for $1 \leqslant i \leqslant n_1$ and $1 \leqslant j \leqslant n_2$, where n_1 is the length of S_1 and n_2 is the length of S_2.

The largest value of suffix similarity is the total weight or score of an optimal local alignment of the two sequences, and the actual local alignment can then be obtained by tracing the dynamic programming table from each such largest suffix similarity value back to the first entry equal to zero.

The actual values chosen as match, mismatch, and gap score determine the local alignment of two sequences.

For instance, with a match score of 1, a mismatch score of 0, and a gap score of 0, the local alignment corresponds to the longest common gapped subsequence, while with a match score of 1, a mismatch score of $-\infty$, and a gap score of $-\infty$, the local alignment corresponds to the longest common subsequence.

Example

The suffix similarities of the DNA sequences GCTTCCGGCTCGTATAATGTGTGG and TGCTTCTGACTATAATAG given in the following dynamic programming table, for a match score of 3, a mismatch score of -1, and a gap score of -3, contain three local alignments of the largest total score, 28.

Each such local alignment can be obtained by following a path of shaded entries (shown here in one case only, for clarity) from the final (largest suffix similarity) back to an initial (zero suffix similarity) entry of the following dynamic programming table, inserting a dash into S_1 when moving to the left and inserting a dash into S_2 when moving up.

In this way, the following local alignment of the two sequences is obtained.

```
GCTTCCGGCTCGTATAAT
||||  |*|  *|  ||||||
GCTT-CTG-AC-TATAAT
```

Example

			T	G	C	T	T	C	T	G	A	C	T	A	T	A	A	T	A	G
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	1	0	0	3	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	3
C	2	0	0	0	6	3	0	3	0	0	2	3	0	0	0	0	0	0	0	0
T	3	0	3	0	3	9	6	3	6	3	0	1	6	3	3	0	0	3	0	0
T	4	0	3	2	0	6	12	9	6	5	2	0	4	5	6	3	0	3	2	0
C	5	0	0	2	5	3	9	15	12	9	6	5	2	3	4	5	2	0	2	1
C	6	0	0	0	5	4	6	12	14	11	8	9	6	3	2	3	4	1	0	1
G	7	0	0	3	2	4	3	9	11	17	14	11	8	5	2	1	2	3	0	3
G	8	0	0	3	2	1	3	6	8	14	16	13	10	7	4	1	0	1	2	3
C	9	0	0	0	6	3	0	6	5	11	13	19	16	13	10	7	4	1	0	1
T	10	0	3	0	3	9	6	3	9	8	10	16	22	19	16	13	10	7	4	1
C	11	0	0	2	3	6	8	9	6	8	7	13	19	21	18	15	12	9	6	3
G	12	0	0	3	1	3	5	7	8	9	7	10	16	18	20	17	14	11	8	9
T	13	0	3	0	2	4	6	4	10	7	8	7	13	15	21	19	16	17	14	11
A	14	0	0	2	0	1	3	5	7	9	10	7	10	16	18	24	22	19	20	17
T	15	0	3	0	1	3	4	2	8	6	8	9	10	13	19	21	23	25	22	19
A	16	0	0	2	0	0	2	3	5	7	9	7	8	13	16	22	24	22	28	25
A	17	0	0	0	1	0	0	1	2	4	10	8	6	11	13	19	25	23	25	27
T	18	0	3	0	0	4	3	0	4	1	7	9	11	8	14	16	22	28	25	24
G	19	0	0	6	3	1	3	2	1	7	4	6	8	10	11	13	19	25	27	28
T	20	0	3	3	5	6	4	2	5	4	6	3	9	7	13	10	16	22	24	26
G	21	0	0	6	3	4	5	3	2	8	5	5	6	8	10	12	13	19	21	27
T	22	0	3	3	5	6	7	4	6	5	7	4	8	5	11	9	11	16	18	24
G	23	0	0	6	3	4	5	6	3	9	6	6	5	7	8	10	8	13	15	21
G	24	0	0	3	5	2	3	4	5	6	8	5	5	4	6	7	9	10	12	18

Algorithm

The dynamic programming table S is filled in for each $0 \leqslant i \leqslant n_1$ and $0 \leqslant j \leqslant n_2$, and the local alignments are obtained by tracing the dynamic programming table from each entry $S[i,j]$ with largest suffix similarity back to a zero suffix similarity entry.

```
procedure local_alignment($S_1, S_2, match, mismatch, gap$)
    $n_1 \leftarrow$ length($S_1$)
    $n_2 \leftarrow$ length($S_2$)
    for $i \leftarrow 0$ to $n_1$ do
        $S[i,0] \leftarrow 0$
    for $j \leftarrow 1$ to $n_2$ do
        $S[0,j] \leftarrow 0$
```

Algorithm

```
for $i \leftarrow 1$ to $n_1$ do
    for $j \leftarrow 1$ to $n_2$ do
        if $S_1[i] = S_2[j]$ then
            $S[i,j] \leftarrow S[i-1,j-1] + match$
        else
            $S[i,j] \leftarrow S[i-1,j-1] + mismatch$
        $S[i,j] \leftarrow \max(S[i,j], S[i-1,j] + gap, S[i,j-1] + gap, 0)$
for $(i,j)$ in $arg\ max(S)$ do
    $T_1 \leftarrow T_2 \leftarrow$ “”
    while $S[i,j] \neq 0$ do
        if $S_1[i] = S_2[j]$ and $S[i,j] = S[i-1,j-1] + match$ then
            $T_1 \leftarrow S_1[i-1] \cdot T_1$
            $T_2 \leftarrow S_2[j-1] \cdot T_2$
            $i \leftarrow i-1$
            $j \leftarrow j-1$
```

Algorithm

```
        **else if** $S_1[i] \neq S_2[j]$ and $S[i,j] = S[i-1, j-1] + \textit{mismatch}$ **then**
            $T_1 \leftarrow S_1[i-1] \cdot T_1$
            $T_2 \leftarrow S_2[j-1] \cdot T_2$
            $i \leftarrow i-1$
            $j \leftarrow j-1$
        **else if** $S[i,j] = S[i-1, j] + \textit{gap}$ **then**
            $T_1 \leftarrow S_1[i-1] \cdot T_1$
            $T_2 \leftarrow$ “–” $\cdot T_2$
            $i \leftarrow i-1$
        **else if** $S[i,j] = S[i, j-1] + \textit{gap}$ **then**
            $T_1 \leftarrow$ “–” $\cdot T_1$
            $T_2 \leftarrow S_2[j-1] \cdot T_2$
            $j \leftarrow j-1$
    **output** $(T_1, T_2)$
```

Remark

The gaps introduced by insertions and deletions in the local alignment of two sequences may be scattered through the sequences, but they may also stick together, forming long runs of consecutive gaps.

The distribution of gaps in the local alignment of two sequences can be influenced by distinguishing between gap opening and gap extension scores, where the combined weight of k consecutive gaps is equal to the gap opening score plus k times the gap extension score.