

Bioinformatics Algorithms

(Fundamental Algorithms, module 2)

Zsuzsanna Lipták

Masters in Medical Bioinformatics
academic year 2017/18, spring term

Pairwise Alignment 2

Semiglobal alignment

match: 1,
mismatch: -1,
gap: -1

```
CAGCGTACACT
---CCTA-----
-----
score -5
```

```
CAGCGTACACT
C--C-T--A--
-----
score -3
```

Semiglobal alignment

match: 1,
mismatch: -1,
gap: -1

```
CAGCGTACT
---CCTA----
```

score -5

```
CAGCGTACT
C--C-T--A--
```

score -3

- The left alignment seems better, but it has a lower score.
- We would like the extremal gaps (before and after the second string) not to count at all.
- Note that this is not covered by local alignment ([why?](#)).

Semiglobal alignment

match: 1,
mismatch: -1,
gap: -1

If we do not count the extremal gaps, then we get:

$$\begin{array}{r} \text{CAGCGTACT} \\ \text{---CCTA----} \\ \hline \text{score } 2 \end{array}$$

$$\begin{array}{r} \text{CAGCGTACT} \\ \text{C--C-T--A--} \\ \hline \text{score } -1 \end{array}$$

... as desired, the score now reflects that the left alignment is better than the right one.

Semiglobal alignment: algorithm

gaps matched here should be free

beginning of s

end of s

beginning of t

end of t

action

0s in first column

maximize over last column

0s in first row

maximize over last row

Semiglobal alignment: algorithm

gaps matched here should be free

beginning of s

end of s

beginning of t

end of t

action

0s in first column

maximize over last column

0s in first row

maximize over last row

Analysis

time and space $O(nm)$

Semiglobal alignment: example

The global similarity of the two strings $s = \text{ACGC}$ and $t = \text{GCTC}$ is 0, with (unique) optimal alignment $\begin{pmatrix} \text{ACGC} \\ \text{GCTC} \end{pmatrix}$. Let us compute an optimal **semiglobal** alignment of s and t , where we set all four types of external gaps as free, and match: $+1$, mism., gap $= -1$.

$D(i,j)$		G	C	T	C
	0	0	0	0	0
A	1	0	-1	-1	-1
C	2	0	-1	0	-1
G	3	0	1	0	-1
C	4	0	0	2	1

optimal
semiglobal
alignment:

ACGC--
--GCTC

score = 2

Semiglobal alignment

N.B.

- Semiglobal alignment is also called *end-space-free alignment*.
- It is not *one* algorithm, but (strictly speaking) 15 different ones, depending on where we want to have charge-free gaps (e.g. beginning and end of first sequence; beginning of first, end of second; etc.)

Applications include:

- find a prefix of s with maximum similarity to t - *which variant do we need?*
- overlap finding (e.g. for sequence assembly): find prefix s' of s and suffix t' of t s.t. $\text{sim}(s', t')$ maximal, or vice versa (prefix of t with suffix of s) - *which variant do we need?*
- substring match: find a substring s' of s with $\text{sim}(s', t)$ maximal - *which variant do we need?*

Affine gap functions

match: 2, mismatch: -1, gap: -1

GACGCTGCCAC
-AC-----CA-

GACGCTGCCAC
-A--C--C-A-

- Both alignments have score 1, but there is a big difference:

Affine gap functions

match: 2, mismatch: -1, gap: -1

GACGCTGCCAC
-AC-----CA-

GACGCTGCCAC
-A--C--C-A-

- Both alignments have score 1, but there is a big difference:
- Assuming that t is similar to a substring of s (namely to ACGCTGCCA), then the first alignment has only one long gap, while the second has 3.

Affine gap functions

match: 2, mismatch: -1, gap: -1

GACGCTGCCAC
-AC-----CA-

GACGCTGCCAC
-A--C--C-A-

- Both alignments have score 1, but there is a big difference:
- Assuming that t is similar to a substring of s (namely to ACGCTGCCA), then the first alignment has only one long gap, while the second has 3.
- Each gap, independent of its length, suggests that **one evolutionary event** happened (insertion or deletion of a stretch of DNA).

Affine gap functions

match: 2, mismatch: -1, gap: -1

GACGCTGCCAC
-AC-----CA-

GACGCTGCCAC
-A--C--C-A-

- Both alignments have score 1, but there is a big difference:
- Assuming that t is similar to a substring of s (namely to ACGCTGCCA), then the first alignment has only one long gap, while the second has 3.
- Each gap, independent of its length, suggests that **one evolutionary event** happened (insertion or deletion of a stretch of DNA).
- The first alignment has one such event, the second three.

Affine gap functions

match: 2, mismatch: -1, gap: -1

GACGCTGCCAC
-AC-----CA-

GACGCTGCCAC
-A--C--C-A-

- Both alignments have score 1, but there is a big difference:
- Assuming that t is similar to a substring of s (namely to ACGCTGCCA), then the first alignment has only one long gap, while the second has 3.
- Each gap, independent of its length, suggests that **one evolutionary event** happened (insertion or deletion of a stretch of DNA).
- The first alignment has one such event, the second three.
- We believe that the first one is more likely (Occam's razor), so should have higher score.

Affine gap functions

match: 2, mismatch: -1, gap: -1

GACGCTGCCAC
-AC-----CA-

GACGCTGCCAC
-A--C--C-A-

- Both alignments have score 1, but there is a big difference:
- Assuming that t is similar to a substring of s (namely to ACGCTGCCA), then the first alignment has only one long gap, while the second has 3.
- Each gap, independent of its length, suggests that **one evolutionary event** happened (insertion or deletion of a stretch of DNA).
- The first alignment has one such event, the second three.
- We believe that the first one is more likely (Occam's razor), so should have higher score.
- **Occam's razor:** The simplest explanation is the best.

Affine gap functions

- We would like to give k gaps in one block a higher score than k individual gaps.
- Longer gaps should have lower score than shorter gaps.

Affine gap functions

- We would like to give k gaps in one block a higher score than k individual gaps.
- Longer gaps should have lower score than shorter gaps.

Affine gap functions:

- gap open: $h < 0$
- gap extend: $g < 0$
- score of k gaps = $h + kg$, for $k \geq 1$
- typically: $h < g$

Affine gap functions

match: 2, mismatch: -1, gaps: $h = -3, g = -1$

GACGCTGCCAC
-AC-----CA-

score = -8

GACGCTGCCAC
-A--C--C-A-

score = -14

Affine gap functions

match: 2, mismatch: -1, gaps: $h = -3, g = -1$

GACGCTGCCAC

-AC-----CA-

score = -8

GACGCTGCCAC

-A--C--C-A-

score = -14

- So now the score reflects that the first al. is better than the second.

Affine gap functions

match: 2, mismatch: -1, gaps: $h = -3, g = -1$

GACGCTGCCAC

-AC-----CA-

score = -8

GACGCTGCCAC

-A--C--C-A-

score = -14

- So now the score reflects that the first al. is better than the second.
- But how do we compute the new score?

Computation

Recall the **central idea** of the DP-algorithm:

Computation

Recall the **central idea** of the DP-algorithm:

If \mathcal{A} is an alignment and \mathcal{B} is the same al. without the last column, then

- $\text{score}(\mathcal{A}) = \text{score}(\mathcal{B}) + \text{score}(\text{last column})$.
- If \mathcal{A} is optimal, then \mathcal{B} is also optimal.
- There are 3 possibilities for the last column:
 1. last column is $\begin{pmatrix} * \\ * \end{pmatrix}$ (char-char)
 2. last column is $\begin{pmatrix} * \\ - \end{pmatrix}$ (char-gap)
 3. last column is $\begin{pmatrix} - \\ * \end{pmatrix}$ (gap-char)

Computation

Recall the **central idea** of the DP-algorithm:

If \mathcal{A} is an alignment and \mathcal{B} is the same al. without the last column, then

- $\text{score}(\mathcal{A}) = \text{score}(\mathcal{B}) + \text{score}(\text{last column})$.
- If \mathcal{A} is optimal, then \mathcal{B} is also optimal.
- There are 3 possibilities for the last column:
 1. last column is $\begin{pmatrix} * \\ * \end{pmatrix}$ (char-char)
 2. last column is $\begin{pmatrix} * \\ - \end{pmatrix}$ (char-gap)
 3. last column is $\begin{pmatrix} - \\ * \end{pmatrix}$ (gap-char)

The problem now is that in cases 1. and 3., the score of the last column **depends on what comes before!** E.g. with $h = -3, g = -1$, the score of $\begin{pmatrix} A \\ - \end{pmatrix}$ is -1 if preceded by a column of the type $\begin{pmatrix} * \\ - \end{pmatrix}$, and -4 otherwise.

Computation

- So we have to distinguish between different types of \mathcal{B} 's (current alignment without last column), according to what type its last column is.

Computation

- So we have to distinguish between different types of B 's (current alignment without last column), according to what type its last column is.
- We will do this via 3 different matrices, each of size $(n + 1)(m + 1)$:
 - $A(i, j)$ = highest score of an alignment of i -length prefix of s and j -length prefix of t ending with $\begin{pmatrix} s_i \\ t_j \end{pmatrix}$
 - $B(i, j)$ = highest score of an alignment of i -length prefix of s and j -length prefix of t ending with $\begin{pmatrix} - \\ t_j \end{pmatrix}$
 - $C(i, j)$ = highest score of an alignment of i -length prefix of s and j -length prefix of t ending with $\begin{pmatrix} s_i \\ - \end{pmatrix}$

Computation

- So we have to distinguish between different types of B 's (current alignment without last column), according to what type its last column is.
- We will do this via 3 different matrices, each of size $(n + 1)(m + 1)$:
 - $A(i, j)$ = highest score of an alignment of i -length prefix of s and j -length prefix of t ending with $\begin{pmatrix} s_i \\ t_j \end{pmatrix}$
 - $B(i, j)$ = highest score of an alignment of i -length prefix of s and j -length prefix of t ending with $\begin{pmatrix} - \\ t_j \end{pmatrix}$
 - $C(i, j)$ = highest score of an alignment of i -length prefix of s and j -length prefix of t ending with $\begin{pmatrix} s_i \\ - \end{pmatrix}$
- Computation of entries will depend on entries from the other matrices.

Computation

Matrix A: Score of last column does not depend on alignment \mathcal{B}

- for $i = 0$ or $j = 0$: There is no alignment ending with a column $\begin{pmatrix} * \\ * \end{pmatrix}$
- for $i, j > 0$: $A(i, j) = \text{best alignment of any type} + \underbrace{\text{match/mismatch}}_{f(s_i, t_j)}$

Computation

Matrix A: Score of last column does not depend on alignment \mathcal{B}

- for $i = 0$ or $j = 0$: There is no alignment ending with a column $\begin{pmatrix} * \\ * \end{pmatrix}$
- for $i, j > 0$: $A(i, j) = \text{best alignment of any type} + \underbrace{\text{match/mismatch}}_{f(s_i, t_j)}$

Computation of entries:

- $A(i, 0) = A(0, j) = -\infty$ for $i = 1, \dots, n, j = 1, \dots, m$, and $A(0, 0) = 0$ (this is necessary for the recursion)
- for $i, j > 0$: $A(i, j) = \max \begin{cases} A(i-1, j-1) + f(s_i, t_j) \\ B(i-1, j-1) + f(s_i, t_j) \\ C(i-1, j-1) + f(s_i, t_j) \end{cases}$

Computation

Matrix B : Score of last column depends on B

- for $j = 0$: There is no alignment ending with a column $\begin{pmatrix} - \\ * \end{pmatrix}$
- for $i = 0, j > 0$: Score of alignment is score of one gap of length j .
- for $i, j > 0$:

$$B(i, j) = \max \begin{cases} \text{best al. of type B + extend an existing gap} \\ \text{best al. of types A or C + start a new gap} \end{cases}$$

Computation

Matrix B : Score of last column depends on B

- for $j = 0$: There is no alignment ending with a column $\begin{pmatrix} - \\ * \end{pmatrix}$
- for $i = 0, j > 0$: Score of alignment is score of one gap of length j .
- for $i, j > 0$:

$$B(i, j) = \max \begin{cases} \text{best al. of type B + extend an existing gap} \\ \text{best al. of types A or C + start a new gap} \end{cases}$$

Computation of entries:

- $B(i, 0) = -\infty$ for $i = 0, \dots, n$,
- $B(0, j) = h + j \cdot g$ for $j = 1, \dots, m$

- for $i, j > 0$: $B(i, j) = \max \begin{cases} A(i, j - 1) + (h + g) \\ B(i, j - 1) + g \\ C(i, j - 1) + (h + g) \end{cases}$

Computation

Matrix C: Score of last column depends on \mathcal{B}

- for $i = 0$: There is no alignment ending with a column $\binom{*}{-}$
- for $i > 0, j = 0$: Score of alignment is score of one gap of length j .
- for $i, j > 0$:

$$C(i, j) = \max \begin{cases} \text{best al. of type C + extend an existing gap} \\ \text{best al. of types A or B + start a new gap} \end{cases}$$

Computation

Matrix C: Score of last column depends on B

- for $i = 0$: There is no alignment ending with a column $\binom{*}{-}$
- for $i > 0, j = 0$: Score of alignment is score of one gap of length j .
- for $i, j > 0$:

$$C(i, j) = \max \begin{cases} \text{best al. of type C} + \text{extend an existing gap} \\ \text{best al. of types A or B} + \text{start a new gap} \end{cases}$$

Computation of entries:

- $C(0, j) = -\infty$ for $j = 0, \dots, m$,
- $C(i, 0) = h + i \cdot g$ for $i = 1, \dots, n$

- for $i, j > 0$: $C(i, j) = \max \begin{cases} A(i-1, j) + (h + g) \\ B(i-1, j) + (h + g) \\ C(i-1, j) + g \end{cases}$

Analysis

- **Space:** for each matrix: $O(nm)$, so altogether $O(nm)$
- **Time:** Computation of every entry is constant, and there are $3(n+1)(m+1) = O(nm)$ entries, so altogether $O(nm)$.
- **Backtracing:** as before, possibly jumping between different matrices.
Time: $O(\text{length of optimal alignment}) = O(n+m)$
- Thus asymptotically the same time and space complexity as the basic algorithm.
- However, we do pay for the better gap function by increasing both time and space by a factor of 3.
- Affine gap penalties are much more reasonable (realistic, useful) than linear gap penalties, and they are universally applied. (All alignment programs use affine gap functions.)