

# Biosequence Algorithms, Spring 2005

## Lecture 10: Approximate Matching, Local Alignments, and Gaps

Pekka Kilpeläinen  
University of Kuopio  
Department of Computer Science

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.127

### Approximate Pattern Matching

Important generalization of exact matching: locate *similar* occurrences of a pattern (not just exact copies)

A substring  $T'$  of  $T$  is an **approximate occurrence** of  $P$  iff the *similarity* of  $P$  and  $T'$  is at least  $\delta$  (for a given  $\delta$ )

Approximate occurrences of pattern  $P$  in text  $T$  can be computed as a slight variation of (global) alignment:

Apply the previous recurrences (with  $P$  in place of  $S_1$ , and  $T$  in place of  $S_2$ ), but change the base condition for row 0 to

$$V(0, j) = 0 \text{ for all } j$$

~ " $T[1], T[2], \dots, T[j]$  are aligned with spaces for free"  
~ "it doesn't cost (or pay) to slide  $P$  along  $T$ "

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.127

### Finding Approximate Occurrences

Table  $V(i, j)$  can be filled, as before, in time  $\Theta(nm)$

**Theorem 11.6.2** An approximate occurrence of  $P[1 \dots n]$  ends at position  $j$  of  $T$  if and only if

$$V(n, j) \geq \delta \quad (1)$$

If (1) holds, there is a path of backpointers btw cells  $(n, j), \dots, (1, k), (0, k')$  such that  $T[k \dots j]$  is an approximate occurrence of  $P$ .

**Proof** (idea) Show, by induction on the length of the path, that  $V(i, j)$  is the similarity of  $P[1 \dots i]$  and  $T[k \dots j]$ .  $\square$

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.127

### Finding Approximate Occurrences (2)

There can be multiple approximate occurrences of  $P$  (of different length) ending at the same position  $j$  of  $T$ .

The *shortest* ones can be located as follows:

1. Find each column  $j$  on row  $n$  where  $V(n, j) \geq \delta$
2. For each of them, trace backpointers from  $(n, j)$  to row 0, preferring pointers '↑' over '↖', and '↖' over '←'

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.127

### Local Alignment

Sometimes a pair of *local regions* of maximal similarity is more interesting than the *global* similarity of the strings ~>

**Local alignment** (or **local similarity**) problem (*paikallinen rinnastus*):

Given strings  $S_1$  and  $S_2$ , find *substrings*  $\alpha$  of  $S_1$  and  $\beta$  of  $S_2$  of *maximal similarity*

**NB:** Substrings of *minimal edit distance* would be exactly matching substrings (possibly of 1 char only);

Maximizing *similarity* is thus more useful for finding longer areas of high similarity

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.127

### Local vs. Global Alignment?

*Global* alignment is often used to compare members of the same protein family

- often of similar length (e.g., globins)
- for trying to infer evolutionary history

*Local* alignment considered more useful

- for comparing anonymous DNA sequences (where only some sections may be related)
- for comparing proteins from different families, to search for common subunits

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.127

### Computing Local Alignment

The local alignment problem btw strings  $S_1[1 \dots n]$  and  $S_2[1 \dots m]$  can be solved in  $O(nm)$  time (T. Smith & M. Waterman, 1981)

- even though there are  $\Theta(n^2m^2)$  possible pairs of substrings!

Helpful assumption: the similarity of two empty strings is 0

Consider first a restricted version of the problem:

Given indices  $i \leq n$  and  $j \leq m$ , the **local suffix alignment problem** is to find a suffix  $\alpha$  of  $S_1[1 \dots i]$  and a suffix  $\beta$  of  $S_2[1 \dots j]$  of maximal similarity (denoted by  $v(i, j)$ )

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.127

### Example of Local Suffix Alignments

**Example:** Let the scores be  $s(x, y) = 2$  when  $x = y \neq \_$ , and  $s(x, y) = -1$  when  $x \neq y$  (for any  $x, y \in \Sigma \cup \{\_\}$ )

Consider strings

```

1 2 3 4 5 6 7
S1: a b c f d e f
S2: f f f c d e
    
```

Then

$$\begin{aligned}
 v(3, 4) &= 2 & (\alpha = \beta = c) \\
 v(4, 5) &= 1 & (\alpha = cf, \beta = cd) \\
 v(5, 5) &= 3 & (\alpha' = f\_d, \beta = fcd)
 \end{aligned}$$

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.127

## Computing the Optimal Local Alignment Value

Denote the value of an optimal local alignment by  $v^*$

**Theorem 11.7.1**  $v^* = \max\{v(i, j) \mid i \leq n, j \leq m\}$

**Proof.**

(1)  $v^* \geq \max\{v(i, j) \mid i \leq n, j \leq m\}$  (local suffix alignment is a special case of local alignments)

(2) Let  $v^*$  be the similarity of substrings  $\alpha$  and  $\beta$  ending at positions  $i^*$  and  $j^*$ , resp.  $\rightarrow \alpha$  and  $\beta$  are suffixes of  $S_1[1 \dots i^*]$  and  $S_2[1 \dots j^*]$ , resp.

$\rightarrow v^* \leq v(i^*, j^*) \leq \max\{v(i, j) \mid i \leq n, j \leq m\}$

(1) & (2) imply the claim  $\square$

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.9/27

## Reducing Local Alignment to Local Suffix Alignment

**Corollary 11.7.2** If  $v(i^*, j^*) = \max\{v(i, j) \mid i \leq n, j \leq m\}$ , then suffixes  $\alpha$  of  $S_1[1 \dots i^*]$  and  $\beta$  of  $S_2[1 \dots j^*]$  whose similarity is  $v(i^*, j^*)$  form a solution to the local alignment problem

How to find these  $i^*$  and  $j^*$ , and  $\alpha$  and  $\beta$ ?

A: By dynamic programming

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.10/27

## Recurrences for Local Suffix Alignment

**Base cases:**  $v(i, 0) = v(0, j) = 0$

(as the score of an empty suffix, assuming  $s(x, \_) \leq 0$ )

**Inductive case** ( $j, i > 0$ ):

How can suffixes  $\alpha$  of  $S_1[1 \dots i]$  and  $\beta$  of  $S_2[1 \dots j]$  be aligned optimally? Different possibilities:

1. they could be empty  $\rightsquigarrow$  score  $v(i, j) = 0$
2.  $S_1[i]$  against  $S_2[j]$   $\rightsquigarrow$  score  $v(i-1, j-1) + s(S_1[i], S_2[j])$
3.  $S_1[i]$  against a space  $\rightsquigarrow$  score  $v(i-1, j) + s(S_1[i], \_)$
4.  $S_2[j]$  against a space  $\rightsquigarrow$  score  $v(i, j-1) + s(\_, S_2[j])$

The optimum for  $v(i, j)$  is obtained by selecting the maximum of the above possibilities

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.11/27

## Local Alignment Recurrences

$\rightsquigarrow$  for  $i, j > 0$ ,

$$v(i, j) = \max \begin{cases} 0 \\ v(i-1, j-1) + s(S_1[i], S_2[j]) \\ v(i-1, j) + s(S_1[i], \_) \\ v(i, j-1) + s(\_, S_2[j]) \end{cases}$$

A table of  $v(i, j)$  values, with backpointers, can be computed applying the recurrences, in a similar way as before

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.12/27

## Complexity of Local Alignment

Maximum value  $v^*$  is found by going through *all* cells of the table, say, in cell  $(i^*, j^*)$ . Substrings  $\alpha$  and  $\beta$  with similarity  $v^*$  are then found by tracing backpointers from cell  $(i^*, j^*)$  along a path  $(i^*, j^*), \dots, (i', j'), (i_0, j_0)$ , where  $v(i_0, j_0) = 0$

Then  $\alpha = S_1[i' \dots i^*]$  and  $\beta = S_2[j' \dots j^*]$

**Theorem 11.7.4** Local alignment between strings  $S_1[1 \dots n]$  and  $S_2[1 \dots m]$  can be computed in time  $O(nm)$

**Proof.** Table  $v(i, j)$  is filled in constant time per cell

The cell  $(i^*, j^*)$  with an optimal score is found in time  $O(nm)$ , and the traceback for  $(i', j')$  requires at most  $n + m$  steps  $\square$

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.13/27

## Remarks

Instead of a single highest-scoring pair  $(\alpha, \beta)$  of substrings, a number of similar substrings, say with similarity above a given threshold, can be found in a similar manner

Suitable scoring schemes are needed for meaningful local alignments:

- scoring matches with 1 and mismatches/spaces with 0 locates *longest common subsequences*
- penalizing mismatches/spaces with large negative values yields *longest common substrings*
- scoring matrices with a positive average score tend to prefer long alignments, which approach *global* alignments

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.14/27

## Alignments with Gaps

A **gap** (*aukko*) is a maximal consecutive run of spaces in a single string participating in an alignment

Alignments with gaps correspond better to the biological phenomena that we try to model (i.e., likelihood of mutational events needed to transform one sequence into the other)

- a deletion or an insertion of an entire (DNA) substring ( $\sim$  gap) often occurs as a single mutational event
- gaps are sometimes key features for inferring evolutionary history of a set of strings

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.15/27

## • How to Score the Gaps?

Different possibilities to score the gaps of an alignment:

- constant, affine, convex, and arbitrary

A **constant** gap weight is the simplest:

Set  $s(\_, x) = s(x, \_) = 0$  for every char  $x$ , and score each gap by constant  $W_g$  (independent of gap length)

$\rightsquigarrow$  the score of an alignment is

$$\sum_{i=1}^l (s(S'_1[i], S'_2[i])) - G \times W_g,$$

where  $S'_1$  and  $S'_2$  are the strings padded with spaces for the alignment, and  $G$  is the total number of gaps

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.16/27

## Affine Gap Weights

Generalization: Treat  $W_g$  as a *gap initiation weight*, and add a *gap extension weight*  $W_s$  for each space

↪ a gap of length  $l$  adds cost  $W_g + W_s \times l$  to the score (which is an “affine” function)

**Affine gap weights** are probably the most commonly used ones in molecular biology

- Default weights of FASTA are  $W_g = 10$  and  $W_s = 2$

Optimal alignments under this model maximize

$$\sum_{i=1}^l (s(S_1'[i], S_2'[i])) - G \times W_g$$

with scores  $s(\_, x) = s(x, \_) = -W_s$  for each  $x$

BSA Lecture 10: Appr. Matching, Local Alignments etc – p.1727

## Convex Gap Weights

Empirical data suggests gap penalties of the form  $W_g + W_s \times \log l$  for gaps of length  $l$

- example of **convex** (*ylöspäin*) *kupera* gap weights, where additional spaces cost less than earlier ones

Finally, **arbitrary** gap weights are unrestricted functions  $w(l)$  of the length of the gap

BSA Lecture 10: Appr. Matching, Local Alignments etc – p.1827

## Time Bounds for Different Gap Weights

Optimal alignments can be found in the following times:

1.  $O(nm^2 + n^2m)$  for arbitrary gap weights
2.  $O(nm \log m)$  for convex gap weights
3.  $O(nm)$  for affine and constant gap weights

We'll discuss the first and the third case in detail

(The algorithm for convex gap weights is more complicated)

BSA Lecture 10: Appr. Matching, Local Alignments etc – p.1927

## Computing Arbitrary Gap-Weight Alignments

(Needleman & Wunsch, 1970)

Consider an optimal alignment between the prefixes  $S_1[1 \dots i]$  of  $S_1$  and  $S_2[1 \dots j]$  of  $S_2$ ; It can either

- align  $S_1[i]$  to the left of  $S_2[j]$  (case *E*)
- align  $S_2[j]$  to the left of  $S_1[i]$ , or (case *F*)
- align  $S_1[i]$  against  $S_2[j]$  (case *G*)

Let  $E_{ij}$  be the maximum value of alignments of type *E*, and respectively  $F_{ij}$  and  $G_{ij}$  the maximum values of alignments of type *F* and *G*.

The maximum value  $V_{ij}$  of any alignment between  $S_1[1 \dots i]$  and  $S_2[1 \dots j]$  is then  $\max\{E_{ij}, F_{ij}, G_{ij}\}$

BSA Lecture 10: Appr. Matching, Local Alignments etc – p.2027

## Recurrences for Arbitrary Gap Weights

Let  $w(l)$  be the weight of a gap of length  $l$

**Base cases:**

$$V_{i,0} = -w(i)$$

$$V_{0,j} = -w(j)$$

(as the cost of aligning a non-empty string with a gap)

Optimums of the different cases for  $i, j > 0$  are as follows:

$$E_{ij} = \max\{V_{i,k} - w(j - k) \mid 0 \leq k < j\}$$

$$F_{ij} = \max\{V_{k,j} - w(i - k) \mid 0 \leq k < i\}$$

$$G_{ij} = V_{i-1,j-1} + s(S_1[i], S_2[j])$$

BSA Lecture 10: Appr. Matching, Local Alignments etc – p.2127

## Complexity

The optimal alignment value  $V_{nm}$  can be computed by filling an  $(n + 1) \times (m + 1)$  table  $V_{ij}$  according to the recurrences

**Theorem 11.8.1** The similarity of  $S_1[1 \dots n]$  and  $S_2[1 \dots m]$  under arbitrary gap weights can be computed in time  $O(nm^2 + n^2m)$

**Proof.** Each  $E_{ij}$  computed by examining  $j$  cells of table  $V$  ↪  $\sum_{j=1}^m j = O(m^2)$  for a single row ↪  $O(nm^2)$  for all  $E_{ij}$

Similarly, each  $F_{ij}$  is computed from  $i$  cells of table  $V$  ↪  $O(mn^2)$  time to compute all values  $F_{ij}$

In addition to that, each of  $V_{ij}$  and  $G_{ij}$  are assigned in constant time □

BSA Lecture 10: Appr. Matching, Local Alignments etc – p.2227

## Affine Gap Weights

Optimal alignments with *affine* gap weights can be computed more efficiently

The reason is that the cost of extending a gap of length  $l$  by one space is now predictable:

$$w(l + 1) = W_g + W_s \times (l + 1) = w(l) + W_s$$

All that matters is whether a new gap is started (with initiation weight  $W_g$ ) or whether it has already begun

This insight is formalized in the recurrences for cases *E* and *F* (using variables  $V_{ij}$ ,  $E_{ij}$ ,  $F_{ij}$  and  $G_{ij}$  in similar roles as before)

BSA Lecture 10: Appr. Matching, Local Alignments etc – p.2327

## Recurrences for Affine Gap Weights

**Base cases:**

$$V_{0,0} = 0$$

$$V_{i,0} = E_{i,0} = -W_g - iW_s$$

$$V_{0,j} = F_{0,j} = -W_g - jW_s$$

(start a gap and make it  $i$  or  $j$  spaces long)

For  $i, j > 0$ ,  $V_{ij} = \max\{E_{ij}, F_{ij}, G_{ij}\}$ , as above

Case *G* of aligning  $S_1[i]$  with  $S_2[j]$  also remains the same:

$$G_{ij} = V_{i-1,j-1} + s(S_1[i], S_2[j])$$

What about cases *E* and *F* (either string ends with a gap)?

BSA Lecture 10: Appr. Matching, Local Alignments etc – p.2427

## Affine Gap Weight Recurrences (2)

Consider case  $E$ , where  $S_1[1 \dots i]$  ends with a gap when aligned with  $S_2[1 \dots j]$

(a) If the gap begins in  $S_1$  opposite  $S_2[j]$ ,  
$$E_{ij} = V_{i,j-1} - W_g - W_s$$

(b) If the gap has begun to the left of  $S_2[j]$ , charge it for the additional space only  $\rightsquigarrow E_{ij} = E_{i,j-1} - W_s$

Whichever the case,  $E_{ij}$  is by definition the maximum:

$$E_{ij} = \max\{E_{i,j-1}, V_{i,j-1} - W_g\} - W_s$$

With similar reasoning

$$F_{ij} = \max\{F_{i-1,j}, V_{i-1,j} - W_g\} - W_s$$

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.25/27

## Time Analysis

As before, the optimal alignment value is found in cell  $V_{n,m}$

**Theorem 11.8.2** The similarity of strings  $S_1[1 \dots n]$  and  $S_2[1 \dots m]$  with affine gap weights can be computed in time  $O(nm)$

**Proof.** The number of values  $V_{ij}$ ,  $E_{ij}$ ,  $F_{ij}$ , and  $G_{ij}$  is  $O(nm)$ , and each of them is computed from a constant number of previously computed values  $\square$

**NB:** The above method computes also similarity with *constant* gap weights, as a special case  $W_s = 0$

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.26/27

## Final Remarks

The *space* required by the tables is more often a problem than the time of computation

For example,  $30,000^2 \text{ B} \approx 1 \text{ GB}$  is much, but  $30,000^2$  operations with  $1/\mu\text{s} \rightsquigarrow \sim 1000 \text{ s}$

It is easy to compute the *score* of an optimal alignment in linear space  $O(\min\{n, m\})$ , by maintaining only a few columns or rows ( $\rightarrow$  exercise)

Hirschberg (1977) has developed an ingenious divide-and-conquer method that also *constructs an optimal alignment* using linear space only (and  $O(mn)$  time)

BSA Lecture 10: Appr. Matching, Local Alignments etc - p.27/27