

---

# Multiple Sequence Alignment

---

- Problem definition and applications
- SP alignment, star alignment and tree alignment
- Hardness of approximation
- Dynamic programming
- Factor 2 approximation for tree alignment

**Martin Zachariasen, DIKU**

February 11, 2004

## Multiple sequence alignment: Problem definition

---

**Given:** A set of input sequences  $s_1, s_2, \dots, s_n$  over a fixed alphabet  $\Sigma$  (e.g. DNA bases or amino acids).

**Task:** Insert spaces into sequences so that the resulting sequences  $s'_1, s'_2, \dots, s'_n$  have the same length and some **cost function is minimized (or score function is maximized)**.

Customary to place all extended strings on top of each other, so that letters in corresponding positions occupy the same column.

**Remark:** Gap/space symbol is denoted by '-'

## Multiple sequence alignment: Applications

---

Functional biological sequences come in *families*.

Multiple sequence alignment can help

- to identify similar parts of a family of sequences
- to identify relationship of an individual sequence to a sequence family
- when making **structural** alignment (folding)
- when making **evolutionary** alignment (phylogeny reconstruction)

One major obstacle: Only core structures are conserved and can be meaningfully alignable; thus some (unknown) positions are more conserved (i.e., should ideally have a higher weight) than others.

## Cost function

---

$s'_1, s'_2, \dots, s'_n$ : set of *aligned* sequences (all of length  $l$ )

$s'_j[i]$ :  $i$ -th letter of string  $s'_j$

We restrict ourselves to purely **additive** cost functions: The cost of an alignment is

$$\sum_{i=1}^l \mu(s'_1[i], s'_2[i], \dots, s'_n[i])$$

where  $\mu : (\Sigma \cup \{-\})^n \mapsto \mathbb{N}$  is the cost of a **column** in the alignment.

Could define  $\mu$  as an  $n$ -dimensional array with  $(|\Sigma| + 1)^n$  entries. Not practical. Need implicit definition.

**Cost scheme**: Specifies the dissimilarity  $\mu(a, b)$  between every pair of letters  $a, b \in \Sigma \cup \{-\}$ .

## Sum-of-pairs (SP) alignment

---

The cost of a column is

$$\mu(s'_1[i], s'_2[i], \dots, s'_n[i]) = \sum_{1 \leq p < q \leq n} \mu(s'_p[i], s'_q[i])$$

Studied and used intensively.

- **Advantages:** Completely symmetrical, each sequence is assumed to be equally related to all other sequences.

Appropriate if all sequences are closely related or relationship is unknown.

- **Disadvantages:** Not appropriate if there are both closely related and remotely related sequences (such as in phylogeny reconstruction).

No probabilistic justification; each sequence is scored as descended from all other sequence instead of a single ancestor (evolutionary events are over-counted).

Costs of individual mismatches decrease as the number of sequences increases.

Note that if  $\mu(-, -) = 0$ , SP cost is equal to sum of pairwise **induced** alignments.

## Star/consensus alignment

---

The cost of a column is

$$\mu(s'_1[i], s'_2[i], \dots, s'_n[i]) = \min_{c \in \Sigma \cup \{-\}} \sum_{j=1}^n \mu(s'_j[i], c)$$

Thus we need to reconstruct a consensus sequence such that its total pairwise cost to the given sequences is minimized.

Consensus sequence can be used as representative for family of input sequences.

## Tree alignment

---

Generalization of star alignment: We are given a tree  $T = (V, E)$  where leaf  $j$  of  $T$  corresponds to input sequence  $s_j$ . Let  $n + 1, n + 2, \dots, n + k$  denote the internal nodes of  $T$ .

The cost of a column is

$$\mu(s'_1[i], s'_2[i], \dots, s'_n[i]) = \min \sum_{(p,q) \in E} \mu(s'_p[i], s'_q[i])$$

where the minimum is taken over all possible assignments of letters to the internal nodes of  $T$ .

Minimum cost for a column can be computed in  $O(n|\Sigma|)$  time.

The tree  $T$  is usually a (rooted) *binary* tree.

## Tree alignment: Alternative formulation

---

**Given:**  $n$  input sequences and a tree  $T$  with  $n$  leaves, each of which is labeled with a unique input sequence.

**Task:** Find an assignment of sequences to the internal nodes of  $T$  such that the total cost of  $T$  is minimized.

The cost of a labeled tree is the total cost of its **edges**; the cost of an edge is equal to the cost of an optimal **pairwise** alignment of its endpoints.

## Hardness results

---

Remark: Some results hold even for *metric* or *simple* (match=0, mismatch=1) pairwise cost schemes.

- SP alignment is NP-hard for  $|\Sigma| = 2$  and metric cost scheme.
- Star alignment NP-hard for  $|\Sigma| = 4$  and simple cost scheme.
- Star alignment MAX SNP-hard for arbitrary pairwise cost scheme.
- Tree alignment NP-hard for  $T$  being a binary tree,  $|\Sigma| = 4$  and metric cost scheme.

## Dynamic programming

---

Generalize dynamic programming for pairwise alignment.

Set of input sequences  $s_1, s_2, \dots, s_n$  (all of length  $m$ ).

$s_j[1, i_j]$ : prefix of  $s_j$  containing the first  $i_j$  letters.

$\Delta = (\Delta_1, \Delta_2, \dots, \Delta_n)$  where  $\Delta_j \in \{0, 1\}$  and  $\Delta \neq 0$ .

Define

$$\Delta_j \cdot s_j[i_j] = \begin{cases} s_j[i_j] & \text{if } \Delta_j = 1 \\ - & \text{if } \Delta_j = 0 \end{cases}$$

## Dynamic programming: Recursion

---

Define  $d(i_1, i_2, \dots, i_n)$  to be the *cost* of an optimal alignment of the prefixes  $s_1[1, i_1], \dots, s_n[1, i_n]$ , where  $0 \leq i_j \leq m$ .

Now we have

$$d(0, \dots, 0) = 0$$

and

$$d(i_1, i_2, \dots, i_n) = \min_{\Delta} \{ d(i_1 - \Delta_1, i_2 - \Delta_2, \dots, i_n - \Delta_n) + \mu(\Delta_1 \cdot s_1[i_1], \Delta_2 \cdot s_2[i_2], \dots, \Delta_n \cdot s_n[i_n]) \}$$

where  $\Delta$  iterates over all valid 0-1 combinations.

$\Delta$  iterates over up to  $2^n - 1$  different values.

$n$ -dimensional table  $d$  contains  $(m + 1)^n$  entries.

Total running time:  $O(2^n(m + 1)^n)$ . Exponential in the number of input strings  $n$ , but polynomial in maximal string length  $m$  for any fixed number of input strings.

## Dynamic programming: Reducing the running time

---

Heuristic method for reducing the number of entries in the dynamic programming table that need to be considered (for SP alignment).

UB: upper bound on the cost of a multiple alignment (i.e., the cost of any heuristic multiple alignment)

$C_{ij}$ : cost of an optimal *pairwise* alignment between  $s_i$  and  $s_j$

Remark:  $C$  and  $C_{ij}$  can be computed quickly in a preprocessing phase.

$C^*$ : cost of an optimal multiple alignment

$C_{ij}^*$ : cost of *induced* pairwise alignment between  $s_i$  and  $s_j$  in an optimal *multiple* alignment

Now we have for any fixed  $x$  and  $y$  ( $1 \leq x < y \leq n$ ):

$$\text{UB} \geq C^* = \sum_{i < j} C_{ij}^* \geq C_{xy}^* - C_{xy} + \sum_{i < j} C_{ij}$$

Thus we have:

$$C_{xy}^* \leq \text{UB}_{xy} := C_{xy} + (\text{UB} - \sum_{i < j} C_{ij})$$

## Dynamic programming: Reducing the running time (cont.)

---

In practice,  $UB_{xy}$  can be used to avoid the processing of many entries in the dynamic programming table.

Let  $C_{xy}[i_x, i_y]$  denote the cost of an *optimal* pairwise alignment in which prefixes  $s_x[1, i_x]$  and  $s_y[1, i_y]$  are forced to be aligned (can also be computed in a preprocessing phase).

If we have

$$C_{xy}[i_x, i_y] > UB_{xy}$$

then all table entries of the form  $d(i_1, \dots, i_x, \dots, i_y, \dots, i_n)$  can be *ignored*.

## Factor 2 approximation for tree alignment

---

Assume that  $T$  is a full and ordered binary tree.

**Lifted tree:** The sequence assigned to an internal node  $v$  equals the sequence of some child of  $v$ .

**Uniformly lifted tree:** Lifted tree in which the *same child* (left or right) is chosen on *each level* of the tree.

### Theorem

There exists a uniformly lifted tree for  $T$  with a cost at most twice from optimum.

Algorithm to find a best uniformly lifted tree: Simple bottom-up dynamic programming in  $T$ . Careful implementation with running time of  $O(m^2nd)$ , where  $d$  is the depth of  $T$ .