

Biosequence Algorithms, Spring 2005

Lecture 12: Computing Multiple String Alignments

Pekka Kilpeläinen
University of Kuopio
Department of Computer Science

Multiple String Alignment

Overview of this section:

1. Sum-of-pairs multiple alignment
 - Exponential-time exact solution
 - Polynomial-time bounded-error approximation
2. Practical approaches (heuristics)

Computing Multiple Alignments

Consider computing *optimal* multiple-string alignments
What is the score or goodness to be optimized?

There is *no universally accepted way to score* multiple alignments

- Many methods in practice compute multiple alignments without any formal guarantee of their quality!

The **sum of pairs score** (*parisummapisteytyys*) is one simple and often used measure for the quality of multiple alignments

- applied, e.g., in a subtask of the MACAW multiple alignment program

The Sum-of-Pairs Score

The sum of pairs score is defined as follows:

The **induced pairwise alignment** of two strings S_i and S_j participating in an alignment \mathcal{M} is obtained by deleting all rows of \mathcal{M} except the rows of S_i and S_j

The **score** of this induced alignment is the two-string alignment score btw the rows of S_i and S_j

- (the score of a space/space alignment is taken to be 0)

The **sum of pairs (SP)** score of a multiple alignment \mathcal{M} is the sum of all pairwise alignment scores induced by \mathcal{M}

Example of the SP Score

Example: Consider a multiple alignment \mathcal{M} of strings $S_1 = AAGAAA$, $S_2 = ATAATG$, and $S_3 = CTGGG$:

$$\begin{array}{l} S'_1 : A A G A A _ A \\ S'_2 : A T _ A A T G \\ S'_3 : C T G _ G _ G \end{array}$$

With standard edit distance the SP score of \mathcal{M} is

$$D(S'_1, S'_2) + D(S'_1, S'_3) + D(S'_2, S'_3) = 4 + 5 + 5$$

- The results to be discussed apply (*weighted*) edit distance as the two-string alignment score, and thus *minimize* the score

Solving the SP Alignment Problem

Given a set of strings $\{S_1, \dots, S_k\}$, the **SP alignment problem** is to compute a global alignment that has a minimal sum-of-pairs score

The SP alignment problem can be solved exactly with dynamic programming, in time $\Theta(2^k n^k)$ (if $n = |S_1| = \dots = |S_k|$)

~ impractical for more than 4 protein-length strings (of a few hundred characters)

Exponential time seems inevitable, since the problem is known to be *NP-complete*

Exact Solution for SP Alignment

Let's sketch the main ideas of a dynamic programming algorithm for computing optimal SP alignment of strings $A[1 \dots n_1]$, $B[1 \dots n_2]$, and $C[1 \dots n_3]$

- Gusfield (Sec. 14.6.1) gives a full pseudocode, but without much explanation

To simplify notation, assume that the score of any char/space alignment is d

The method fills an array $D(i, j, k)$ of size $(n_1 + 1) \times (n_2 + 1) \times (n_3 + 1)$, where

- $D(i, j, k)$ is the optimal SP score of aligning the prefixes $A[1 \dots i]$, $B[1 \dots j]$ and $C[1 \dots k]$

SP Alignment Recurrences

Base cases:

$D(0, 0, 0) = 0$ is rather obvious

What about the initial boundaries of $D(i, j, k)$ with $i = 0$, $j = 0$, or $k = 0$?

Consider $D(i, j, 0)$: What is the SP score for an optimal alignment of $A[1 \dots i]$, $B[1 \dots j]$ and $C[1 \dots 0] = \epsilon$?

A: $D_{A,B}(i, j) + (i + j) \times d$, where $D_{A,B}(i, j)$ is the edit distance of $A[1 \dots i]$ and $B[1 \dots j]$

Formulas for $D(i, 0, k)$ and $D(0, j, k)$ are similar

Computing Inner Cells of $D(i, j, k)$

The values of inner cells $D(i, j, k)$ with $i, j, k > 0$ depend on seven adjacent cells $D(i', j', k')$ where $i' = i - 1, j' = j - 1$ or $k' = k - 1$

The optimum for $D(i, j, k)$ is the *minimum* of the seven cases sketched below:

First, an optimal alignment of $A[1 \dots i], B[1 \dots j]$ and $C[1 \dots k]$ can align the last chars of each, giving score

$$D(i-1, j-1, k-1) + s(A[i], B[j]) + s(A[i], C[k]) + s(B[j], C[k])$$

Computing Inner Cells of $D(i, j, k)$

Second, any of the 3 strings can end with a space aligned with the last chars of the other two; This gives a score

$$D(i-1, j-1, k) + s(A[i], B[j]) + 2d$$

(if ' ' was inserted at the end of C ; A and B similarly)

Finally, any two of the strings can end with a space aligned against the last char of the third string, say C, \rightsquigarrow

$$D(i, j, k-1) + 2d$$

Each cell can be filled in constant time according to the recurrences \rightsquigarrow total time is $O(n_1 n_2 n_3)$

Speedup Heuristics

Computation of value $D(n_1, n_2, n_3)$ can be considered also as finding a *shortest path* from cell $(0, 0, 0)$ to (n_1, n_2, n_3)

Standard path finding heuristics (Branch-and-bound, A^*) can be applied to prune the search space (that is, cells evaluated)

Such optimizations are applied in the multiple sequence alignment program called MSA

- reported to align six strings of ~ 200 characters in a "practical" amount of time

A Bounded-Error Approximation

Next: a polynomial-time method for approximating an optimal SP-alignment (Gusfield, 1993)

Gusfield's method (like also some heuristics of practice) is based on extending an alignment by one string at a time

Key idea: When aligning a set of strings $S = \{S_1, \dots, S_k\}$, concentrate to optimizing $k - 1$ pairwise distances given in the form of a *tree*

The tree used in Gusfield's method is a **center star** (to be explained)

Alignments Consistent with a Tree

Let T be a (*free*) *tree* (connected acyclic graph) that has strings S_1, \dots, S_k as its nodes

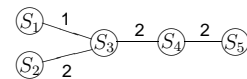
A multiple alignment \mathcal{M} of $S = \{S_1, \dots, S_k\}$ is **consistent with T** if the score of the induced pairwise alignment of S_i and S_j equals $D(S_i, S_j)$ whenever (S_i, S_j) is an edge of T

- That is, strings that are adjacent in the tree are aligned in a pairwise optimal way (while the induced score of other pairs does not matter)

Example of Alignment Consistent with Tree

Consider strings $S_1 = AXZA, S_2 = AXZB, S_3 = AXXZA, S_4 = AYZA, S_5 = AYYXZA$

A tree T of strings S_1, \dots, S_5 (with edit distances on edges):



An alignment consistent with tree T :

```

S1: A X _ _ Z A
S2: A _ X _ Z B
S3: A X X _ Z A
S4: A Y _ _ Z A
S5: A Y X X Z A
    
```

Computing an Alignment Consistent with a Tree

Theorem 14.6.1 Given a set of strings $S = \{S_1, \dots, S_k\}$ and a tree T made of strings of S , we can efficiently compute a multiple alignment \mathcal{M} of S that is consistent with T

Proof. (Sketch)

First compute an optimal alignment (with distance $D(S_i, S_j)$) for any pair of strings S_i and S_j adjacent in the tree

Until all strings have been aligned, select two strings, \bar{S}_h in \mathcal{M} (possibly with inserted spaces) and S' not yet aligned, s.t. (S_h, S') is an edge of T ; Align \bar{S}_h and S' , assigning zero weight for spaces inserted in S' against spaces in \bar{S}_h
 \rightsquigarrow this alignment has score $D(S_h, S')$

Computing a Tree-Based Alignment

Then add \bar{S}' (possibly with added spaces) to the alignment; If new spaces were inserted in \bar{S}_h , add them in corresponding columns of other rows of \mathcal{M} , too
 \rightsquigarrow the induced pairwise scores of \mathcal{M} do not change

Complexity: If l is the final row-length of \mathcal{M} , all the $k - 1$ pairwise alignments can be computed in total time $O(kl^2)$ \square

Center Star Method

Tree-based alignment is applied to approximating an optimal alignment, by using a tree where other strings are connected to a suitably selected *center string*

Define the **consensus error** (*konsensusvirhe*) of a string S relative a set of strings \mathcal{S} to be $E(S) = \sum_{S_i \in \mathcal{S}} D(S, S_i)$

A string $S_c \in \mathcal{S}$ is a **center string** if $E(S_c) \leq E(S)$ for all $S \in \mathcal{S}$

- ⦿ A center string can easily be found in polynomial time

A **center star** is a tree that consists of an edge btw the center string and each other string of \mathcal{S}

BSA Lecture 12: Computing Multiple Alignments – p.17/25

Center-Star Alignment

First, find a center string of $\mathcal{S} = \{S_1, \dots, S_k\}$

Then compute a multiple alignment \mathcal{M}_c consistent with the center star of \mathcal{S} (applying method of Th. 14.6.1)

How good is the result?

Denote the SP-score of alignment \mathcal{M}_c by $d(\mathcal{M}_c)$, and let $d(\mathcal{M}^*)$ be the SP-score of an optimal alignment of \mathcal{S}

Main result: $d(\mathcal{M}_c)$ is less than two times the optimum score $d(\mathcal{M}^*)$:

BSA Lecture 12: Computing Multiple Alignments – p.18/25

Goodness of Center-Star Alignment

Theorem 14.6.2 $d(\mathcal{M}_c) \leq (2 - 2/k) \times d(\mathcal{M}^*)$

Proof. (Omitted; See Gusfield) □

NB: The result is based on assuming that the scoring scheme satisfies the *triangle inequality*:

$$s(x, y) \leq s(x, z) + s(z, y)$$

- ⦿ not all scoring matrices in computational biology satisfy the triangle inequality

BSA Lecture 12: Computing Multiple Alignments – p.19/25

Remarks on Center-Star Approximation

In practice, the SP score $d(\mathcal{M}_c)$ of a center-star alignment *could* be much less than twice the optimal score

- ⦿ in limited tests $d(\mathcal{M}_c)$ deviated from the optimal SP score by 2–16% only

Also a *polynomial time approximation scheme* has been developed for the SP alignment problem

A method of (Bafna, Lawler & Pevzner, 1994) produces a multiple alignment of k strings with SP score $\leq 2 - q/k$ times the optimum;

Accuracy of the approximation can be increased, but that also increases the running time (as a function of q)

BSA Lecture 12: Computing Multiple Alignments – p.20/25

Commonly Applied Heuristics

Numerous multiple alignment methods and programs are used in practice, and results produced by them are reported in hundreds of papers

Bounded-error heuristics are not widely used in practice for multiple alignment

Most methods apply variants of two ideas: **iterative pairwise alignments** and **finding motifs common to the strings**

Methods that apply **iterative pairwise alignments** build an alignment by iteratively merging alignments of two subsets of the strings together

BSA Lecture 12: Computing Multiple Alignments – p.21/25

Iterative Pairwise Alignments

A simple example: Compute the edit distances btw all strings, and begin by aligning the most similar pair of strings

Then repeatedly align the closest pair of strings (S_i, S_j) s.t. one of them is in the alignment and the other is not (as in the proof of Th. 14.6.1)

↪ The method computes an alignment that is consistent with a *minimum spanning tree* (wrt the pairwise edit distances)

BSA Lecture 12: Computing Multiple Alignments – p.22/25

Iterative Alignments and Clustering

Other variants merge larger sub-alignments together

- ⦿ merging could happen by aligning some representation (profile, consensus sequence) of the sub-alignments

Methods can be seen as applications of ideas from *clustering* (*klusterointi, ryvästys*)

Which of the variants are superior is a difficult question to answer

BSA Lecture 12: Computing Multiple Alignments – p.23/25

Repeated-Motif Methods

The second major category of heuristics first find a **motif** common to many strings of \mathcal{S}

- ⦿ a substring or a small similar subsequence
- ⦿ aka *anchor, core, block, region, q-gram* etc

When a “good” motif (wide and common to many strings) is found, strings containing it are shifted s.t. occurrences of motifs are aligned against each other

Then substrings on each side of the motifs are aligned recursively; When no good motifs are found, remaining subproblems are solved by iterative alignment

BSA Lecture 12: Computing Multiple Alignments – p.24/25

Repeated-Motif Methods (2)

Strings not containing motifs are aligned separately, and the two sub-alignments are finally merged together

For example, the MACAW program locates motifs at the top level, but aligns strings between motifs applying SP score

Again, there are numerous ways to realize these ideas, and it is hard to justify which are the best