Special course in Computer Science: Advanced Text Algorithms

Lecture 8: Multiple alignments

Eugen Czeizler
Department of IT, Abo Akademi
http://combio.abo.fi/teaching/textalg/

(slides originally by I. Petre, E. Czeizler, V. Rogojin)

Multiple string comparison

- Multiple alignment is a specific formalization of multiple string comparison, which is one of the most important methodologies and active research areas in bio-sequence analysis.

- It is used for
  - extracting and representing biologically important commonalities from a set of strings, which might go unnoticed if only two strings were compared
  - inferring evolutionary history from DNA or protein sequences

Multiple alignments

- A (global) multiple alignment of k > 2 strings is a natural generalization of two-string alignments:

- Given several sequences, find their “best” simultaneous alignment, find which parts of the sequences are similar and which are not, etc.

Multiple alignment

- Multiple alignment is also used in a fundamental different way than the 2-string comparison: for biological inference problems that are inverse to problems addressed by 2-string alignments.

  - In DB searches, 2-string alignments find strings having in common subpatterns aiming to identify unsuspected biological relashionship

  - Multiple alignments aim to deduce unknown conserved subpatterns from a set of strings already known to be biologically related.

Multiple alignments

- *Example*: having the sequences for certain proteins that have similar functions in a number of different species, find out which parts of the sequences are similar – get a multiple alignment of all the amino acid sequences
  - These similarities may reveal evolutionary history, critical conserved motifs or conserved characters, common 2D and 3D molecular structure, clues about common biological function
  - Useful when working on families of proteins
  - Many such similarities are quite faint and dispersed – they will not be visible when comparing two sequences alone
  - "One or two homologous sequences whisper… a full multiple alignment shouts out loud" (*A.Lesk)*

Multiple alignments

- *The first fact of biological sequence analysis:* *In biomolecular sequences (DNA, RNA, amino acid sequences), high sequence similarity usually implies significant functional or structural similarity*
  - *Underlines the importance of 2-sequence comparison and DB searches*

- *The second fact of biological sequence analysis:* *Evolutionary and functionally related biomolecular sequences can differ significantly throughout much of the sequence and yet preserve the same 3D structure, or the same 2D substructures (motifs), or the same active sites, etc.*

Multiple alignments

- Example: hemoglobin is a nearly universal protein containing 4 chains of about 140 amino acids each; it is contained in organisms as diverse as mammals and insects and its function is essentially the same (it binds and transports oxygen)
  - Pairwise alignment of hemoglobin chains in two mammals will immediately suggest functional similarity (in humans and chimpanzee the sequences are identical)
  - Pairwise alignment of sequences from a human and an insect will reveal very little – on average, 100 mutations in each chain (during 600 million years)

Multiple alignments

- The ability of many proteins to preserve structure of function in the face of massive mutations is an empirical fact although it is quite counterintuitive.

  - There are critical amino acid positions in proteins where mutations cause dysfunctions, e.g., mad cow disease

  - However, on the whole, many such positions are non-critical, and over evolutionary history have sustained many mutations without a destructive effect on the protein.

Multiple alignment

- A global multiple alignment of sequences $S_1,\ldots,S_k$ is obtained by inserting spaces in the sequences so that they all have the same length, say l, and then arrange them in k rows and l columns, each character or space in a unique column
  - When we align them we require that no column consists of spaces only

- Example: A multiple alignment of strings {abca;ababa;accb;cbbc}

$$\begin{array}{ccccc} a & b & c & \_ & a \\ a & b & a & b & a \\ a & c & c & b & \_ \\ c & b & \_ & b & c \end{array}$$

Local multiple alignment

- There are also local multiple alignments.

- A local multiple alignment of strings $S_1,...,S_k$ consists of selecting a substring S'_i of each S_i, and aligning these k substrings globally.

- We restrict to considering global multiple alignments only.

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 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 ignoring all other rows of $\mathcal{M}$ except those of S_i and S_j

- The score of this induced alignment is the two-string alignment score between 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:

$$\begin{array}{lccccccc} S_1': & A & A & G & A & A & \_ & A \\ S_2': & A & T & \_ & A & A & T & G \\ S_3': & C & T & G & \_ & G & \_ & G \end{array}$$

- 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 $\{S_1,...,S_k\}$, the SP alignment problem asks 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 $\Theta(2^k n^k)$ (where $n = |S_1| = ... = |S_k|$)
  - 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-hard

Exact Solution for SP Alignment

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

- Assume that the following weights (scores) for the edit operations:
  - d for deletion/insertion (i.e., align a character with a space),
  - r for substitution (i.e., align two mismatching characters),
  - e for match (i.e., align two matching characters)

- The method fills an array D(i,j,k) of size $(n_1+1)(n_2+1)(n_3+1)$, where D(i,j,k) is the optimal SP score of aligning the prefixes A[1..i], B[1..j] and C[1..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..i], B[1..j] and C[1..0]=ε ?
  - $D(i,j,0)=D_{A,B}(i,j)+(i+j)\times d$, where $D_{A,B}$ is the edit distance between A[1..i] and B[1..j]
- In a similar way we obtain formulas also for D(i,0,k) and D(0,j,k)

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 value for D(i,j,k) is the minimum of the seven cases sketched below:

1) An optimal alignment of A[1..i], B[1..j] and C[1..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)

2) Any of the 3 strings can end with a space aligned with the last chars of the other two.

This gives scores:

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

3) Finally, any two of the strings can end with a space aligned against the last char of the third string.

This gives scores:

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

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

Speedup for the exact solution

- In the dynamic programming approach, when we compute a value D(i,j,k), we have to look back to the (at most) seven cells that influence this value.

- Another approach: after we compute a value D(i,j,k) we will send it forward to the (at most) seven cells whose value is influenced by it.

- Then, the computation of the value $D(n_1,n_2,n_3)$ can be seen as finding a shortest path from cell (0,0,0) to (n_1,n_2,n_3)

Speedup for the exact solution

- Standard path finding heuristics (e.g., branch-and-bound, or 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 other heuristics used in practice) is based on extending an alignment by one string at a time

- Key idea: When aligning a set of k strings $\mathcal{S}=\{S_1,\ldots,S_k\}$, 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 (which will be explained later)

Alignments Consistent with a Tree

- Let T be a tree (i.e., connected acyclic graph) that has strings $S_1,...,S_k$ as its nodes

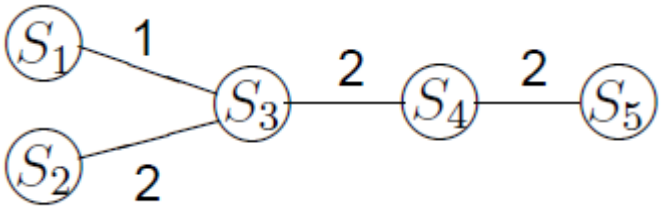
- A multiple alignment $\mathcal{M}$ of $\mathcal{S}=\{S_1,...,S_k\}$ is consistent with T if, for all edges (S_i,S_j) of T, the score of the induced pairwise alignment of S_i and S_j equals $D(S_i,S_j)$

- 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=AYXXZA

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



- An alignment consistent with tree T:

$$
\begin{array}{lcccccc}
S'_1: & A & X & \_ & \_ & Z & A \\
S'_2: & A & \_ & X & \_ & Z & B \\
S'_3: & A & X & X & \_ & Z & A \\
S'_4: & A & Y & \_ & \_ & Z & A \\
S'_5: & A & Y & X & X & Z & A
\end{array}
$$

Computing an Alignment Consistent with a Tree

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

Proof. (Sketch)

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

- Until all strings have been aligned, select two strings, S'_h in $\mathcal{M}$ (possibly with inserted spaces) and S_j not yet aligned, s.t. (S_h,S_j) is an edge of T;

- Align S'_h and S_j, assigning zero weight for spaces inserted in S_j against spaces in S'_h
  - this alignment has score $D(S_h,S_j)$

Computing an Alignment Consistent with a Tree

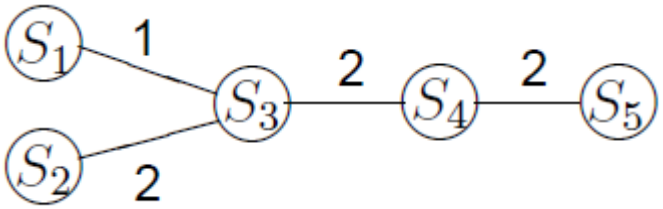
- Then add S_j' (possibly with added spaces) to the alignment;

- If new spaces were inserted in S'_h, add them in corresponding columns of other rows of $\mathcal{M}$, too
  - 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 $O(k\ l^2)$

Example

- Let us take the strings in the previous example: S_1=AXZA, S_2=AXZB, S_3=AXXZA, S_4=AYZA, S_5=AYXXZA and the tree:



- We first chose S_2 and S_3 and compute their optimum alignment

$$S'_2: \quad A \quad \_ \quad X \quad Z \quad B$$
$$S'_3: \quad A \quad X \quad X \quad Z \quad A$$

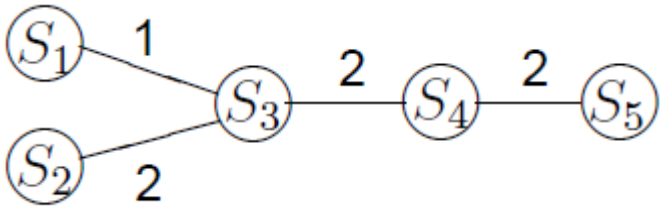
- Then, we chose S'_h=S'_3 which is in the alignment and S_4 which is not but S_3 and S_4 are adjacent in the tree and compute their alignment

$$S'_3: \quad A \quad X \quad X \quad Z \quad A$$
$$S'_4: \quad A \quad Y \quad \_ \quad Z \quad A$$

Example

- Then we add S'_4 to the multiple alignment

$$
\begin{array}{lcccccc}
S'_2: & A & \_ & X & Z & B \\
S'_3: & A & X & X & Z & A \\
S'_4: & A & Y & \_ & Z & A
\end{array}
$$



- Then we select $S'_h=S'_4$ which is in the alignment and S_5 which is not but S_4 and S_5 are adjacent in the tree and compute their alignment

$$
\begin{array}{lcccccc}
S'_4: & A & Y & \_ & \_ & Z & A \\
S'_5: & A & Y & X & X & Z & A
\end{array}
$$

- When we add S'_5 to the multiple alignment we have to include an additional space in all previous strings

$$
\begin{array}{lcccccc}
S'_2: & A & \_ & X & \_ & Z & B \\
S'_3: & A & X & X & \_ & Z & A \\
S'_4: & A & Y & \_ & \_ & Z & A \\
S'_5: & A & Y & X & X & Z & A
\end{array}
$$

Center Star Method

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

- Define the consensus error *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 between the center string and each other string of $\mathcal{S}$

Center-Star Alignment

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

- Then compute a multiple alignment $\mathcal{M}_c$ consistent with the center star of $\mathcal{S}$ (applying method in the previous theorem)

- We consider here an alphabet-weighted scoring scheme, i.e., s(x,y) is the score contributed by aligning character x with character y

- Then $\mathcal{M}_c$ is a reasonable approximation of an optimal alignment, provided that scoring 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

Approaching Optimal Alignment

- Denote by $d_c(S_i,S_j)$ the score of the pairwise alignment of strings S_i and S_j induced by $\mathcal{M}_c$; Call it the distance of S_i and S_j induced by alignment $\mathcal{M}_c$
  - $d_c(S_c,S_i)=D(S_c,S_i)$, $d_c(S_i,S_j)\geq D(S_i,S_j)$,

- Lemma: If the character scores satisfy the triangle inequality, then

$$d_c(S_i,S_j) \leq d_c(S_i,S_c)+d_c(S_c,S_j) \quad (1)$$
$$= D(S_i,S_c)+D(S_c,S_j) \quad (2)$$

for any S_i, $S_j \in \mathcal{S}$

Proof.

- (1) follows because the triangle equality holds at each column of the alignment of S_i, S_c and S_j
- (2) holds because the alignment $\mathcal{M}_c$ is consistent with the center star

Bound to the Error

- Let $\mathcal{M}^*$ be an optimal multiple alignment of $\mathcal{S}$

- Denote by $d^*(S_i,S_j)$ the score of the pairwise alignment of S_i and S_j induced by $\mathcal{M}^*$

- Denote the score of the alignment $\mathcal{M}_c$ by

$$d(\mathcal{M}_c) = \sum_{i<j} d_c(S_i, S_j)$$

and the score of the optimal alignment $\mathcal{M}$ by

$$d(\mathcal{M}^*) = \sum_{i<j} d^*(S_i, S_j)$$

Main result: $d(\mathcal{M}_c) < 2d(\mathcal{M}^*)$

Remarks on Center-Star Approximation

- The 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 polynomial time approximation schemes have been developed for the SP alignment problem
  - The user knows the amount of running time needed to achieve any particular level of confidence.

- A method of Bafna, Lawler & Pevzner, 1994 produces a multiple alignment of k strings with SP score ≤ 2 - q/k times the optimum, for any chosen q;
  - Accuracy of the approximation can be increased, but that also increases the running time (as a function of q)

Commonly Applied Heuristics

- Many multiple alignment methods and programs are used in practice.

- 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 together alignments of two subsets of the strings

Iterative Pairwise Alignments

- A simple example: If we want to compute the edit distances between all strings, then we 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.

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

Iterative Alignments and Clustering

- Other variants merge together larger sub-alignments
  - merging could happen by aligning some representation (profile, consensus sequence) of the sub-alignments

- Methods can be seen as applications of ideas from *clustering.*

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

Repeated-Motif Methods

- The second major category of heuristics first finds a motif common to many strings of 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.

Repeated-Motif Methods

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

- For example, the MACAW program, which is a widely used program, locates motifs at the top level, but aligns strings between motifs applying SP score.

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