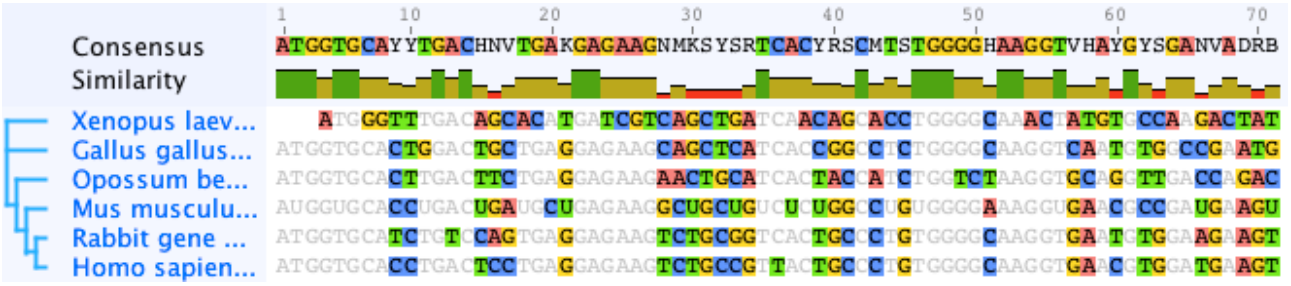
7 Multiple Sequence Alignment

The exposition was prepared by Clemens GrÃPpl, based on earlier versions by Daniel Huson, Knut Reinert, and Gunnar Klau. It is based on the following sources, which are all recommended reading:

1. R. Durbin, S. Eddy, A. Krogh und G. Mitchison: Biological sequence analysis, Cambridge, 1998
2. Gusfield: Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997, chapter 14.
3. J. Stoye, P. Husemann, E. Willing, S. Rahmann, R. Giegerich, S. Kurtz, E. Ohlebusch et al.: Sequence Analysis I + II, lecture notes for WS 2009/10 and SS 2010, Universitï Bielefeld, `http://wiki.techfak.uni-bielefeld.de/gi/GILectures/2009winter/SequenzAnalyse`

7.1 Introduction

What is a multiple sequence alignment? A multiple sequence alignment is an alignment of more than two sequences:



[Sequences are truncated. Screenshot from http://www.geneious.com/]

In this example multiple sequence alignment is applied to a set of sequences that are assumed to be homologous (i. e., having a common ancestor). Homologous nucleotides are placed in the same column of a multiple alignment. Note the phylogenetic tree attached to the left.

While multiple sequence alignment (MSA) is natural generalization of pairwise sequence alignment, there are lots of new questions and issues concerning the scoring function, the significance of scores, the gap penalties, and efficient implementations.

Definitions.
Assume we are given k sequences $x_1, \ldots, x_k$ over an alphabet Σ.

Let $- \notin \Sigma$ be the *gap symbol*. Let $h\colon (\Sigma \cup \{-\})^* \to \Sigma^*$ be the *projection* mapping that removes all gap symbols from a sequence over the alphabet $\Sigma \cup \{-\}$. For example, $h(\texttt{-FPIKWTAPEAALY---GRFT}) = \texttt{FPIKWTAPEAALYGRFT}$.

Then a *global alignment* of $x_1, \ldots, x_k$ consists of k sequences $x'_1, \ldots, x'_k$ over the alphabet $\Sigma \cup \{-\}$ such that

- $h(x'_i) = x_i$ for all i,
- $|x'_i| = |x'_j|$ for all i, j, and
- $(x'_{1,p}, \ldots, x'_{k,p}) \neq (-, \ldots, -)$ for all p.

The concept of *projections* can be extended to subsets of sequences. We remove columns that consist of gap symbols only:

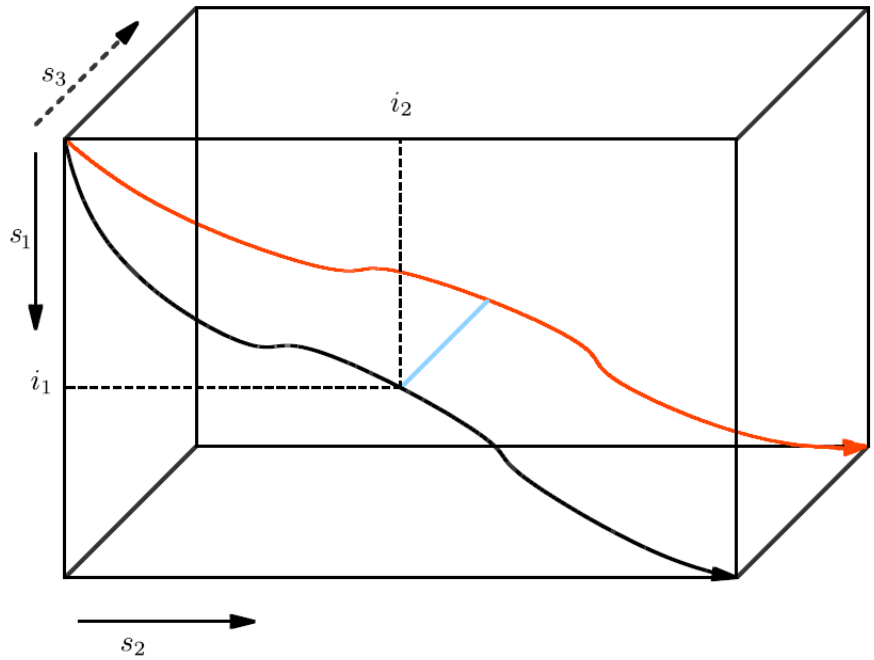
$$A = \begin{pmatrix} - & A & C & C & - & - & A & T & G \\ - & A & - & C & G & A & A & T & - \\ T & A & C & C & - & - & A & G & G \\ - & A & - & C & C & A & A & T & G \end{pmatrix}$$

$$\pi_{\{1,2\}}(A) = \begin{pmatrix} A & C & C & - & - & A & T & G \\ A & - & C & G & A & A & T & - \end{pmatrix}$$

$$\pi_{\{1,3\}}(A) = \begin{pmatrix} - & A & C & C & A & T & G \\ T & A & C & C & A & G & G \end{pmatrix}$$

$$\pi_{\{3,4\}}(A) = \begin{pmatrix} T & A & C & C & - & - & A & G & G \\ - & A & - & C & C & A & A & T & G \end{pmatrix}$$

Another view at the concept of alignment projection:



Projection of a multiple alignment of three sequences s_1, s_2, s_3 on s_1 and s_2.

An incomplete list of the many uses of multiple sequence alignment:

- detecting faint similarities in sequences that are not detected by pairwise sequence comparison.
- detecting structural homologies.
- grouping proteins into families.
- computing the consensus sequence in assembly projects.
- inferring evolutionary trees.
- and more ...

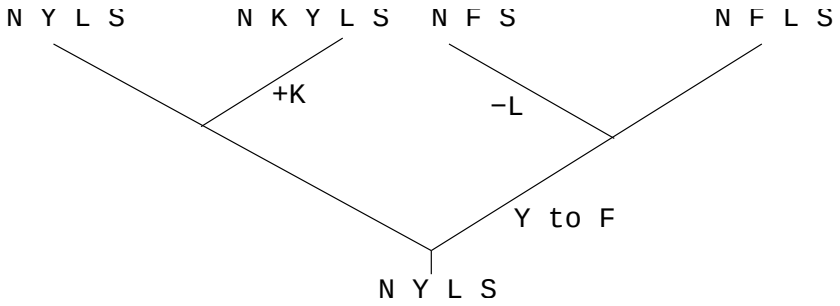
We now give some more details about the different uses.

One or two homologous sequences whisper ... a full multiple alignment shouts out loud. [1]

7.2 MSA and evolutionary trees

One main application of multiple sequence alignment lies in phylogenetic analysis. Given an MSA, we would like to reconstruct the evolutionary tree that gave rise to these sequences, e.g.:

$a_1 =$ N - F L S
$a_2 =$ N - F - S
$a_3 =$ N K Y L S
$a_4 =$ N - Y L S



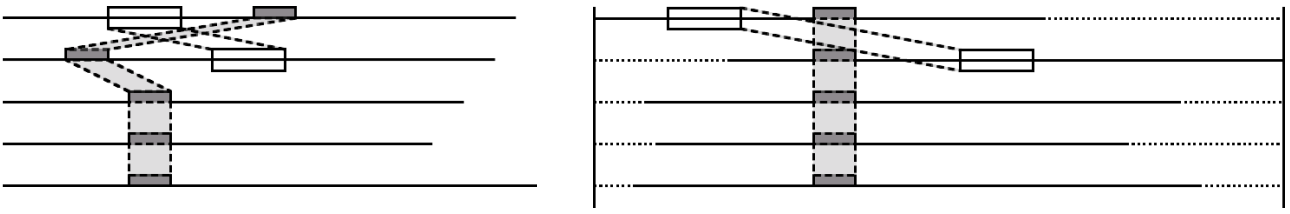
We need to find out how the positions of the sequences correspond to each other. The problem of computing phylogenetic trees (for each column) will be discussed later.

7.3 Protein families

Assume we have established a family $s_1, s_2, \ldots, s_r$ of homologous protein sequences. Does a new sequence s_0 belong to the family?

One method of answering this question would be to align s_0 to each of $s_1, \ldots, s_r$ in turn. If one of these alignments produces a high score, then we may decide that s_0 belongs to the family.

However, perhaps s_0 does not align particularly well to any one specific family member, but does well in a multiple alignment, due to common motifs etc.



7.4 Sequence Assembly

Assume we are given a layout of several genomic reads in a sequencing project that were produced using the shotgun sequencing method. These fragments will be highly similar, and hence easy to align. Nevertheless we

[1] T. J. P. Hubbard, A. M. Lesk, and A. Tramontano. Gathering them into the fold. Nature Structural Biology, 4:313, 1996.

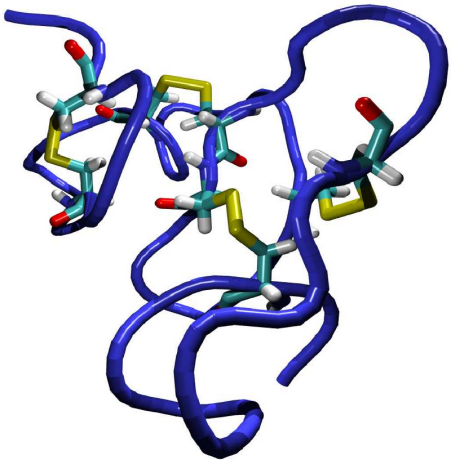
want to do this with great speed and accuracy:

```
f1  ACCACAACCCTGCATGGGGCAT-ATTTGGCCTAGCT
f2                  AGGGCCTTATATG-GCTAGCT-CGTTCCCGGGCATGGC
f3             GCATGGGGCATTATCTGGCCTAGCT--GAT
f4                                      CCGTTCCCGG-CTTGGCAACG
============================================================
cns ACCACAACCCTGCATGGGGCATTATCTGGCCTAGCT-CGTTCCCGGGCATGGCAACG
```

7.5 Conservation of structural elements

The below figure shows the alignment of N-acetylglucosamine-binding proteins and the tertiary structure of one of them, the hevein.

```
AATAHAQRCG EQGSNMECPN NLCCSQYGYC GMGGDYCGKG ..CQNGACYT
VAATNAQTCG KQNDGMICPH NLCCSQFGYC GLGRDYCGTG ..CQSGACCS
VGLVSAQRCG SQGGGGTCPA LWCCSIWGWC GDSEPYCGRT ..CENK.CWS
AATAQAQRCG EQGSNMECPN NLCCSQYGYC GMGGDYCGKG ..CQNGACWT
AATAQAQRCG EQGSNMECPN NLCCSQYGYC GMGGDYCGKG ..CQNGACWT
......QRCG EQGSGMECPN NLCCSQYGYC GMGGDYCGKG ..CQNGACWT
SETVKSQNCG .......CAP NLCCSQFGYC GSTDAYCGTG ..CRSGPCRS
RGSAE..QCG RQAGDALCPG GLCCSSYGWC GTTVDYCGIG ..CQSQ.CDG
AGPAAAQNCG .......CQP NFCCSKFGYC GTTDAYCGDG ..CQSGPCRS
AGPAAAQNCG .......CQP NVCCSKFGYC GTTDEYCGDG ..CQSGPCRS
RGSAE..QCG QQAGDALCPG GLCCSSYGWC GTTADYCGDG ..CQSQ.CDG
RGSAE..QCG RQAGDALCPG GLCCSFYGWC GTTVDYCGDG ..CQSQ.CDG
TGVAIAEQCG RQAGGKLCPN NLCCSQWGWC GSTDEYCSPD HNCQSN.CK.
......EQCG RQAGGKLCPN NLCCSQYGWC GSSDDYCSPS KNCQSN.CK.
```



The example exhibits 8 cysteins that form 4 disulphid bridges (shown yellow in the 3d structure) and are an essential structural part of those proteins.

7.6 The dynamic programming algorithm for MSA

Although local alignments are biologically more relevant, it is easier to discuss global MSA. Dynamic programming algorithms developed for pairwise alignment can be modified to MSA. For concreteness, let us discuss how to compute a global MSA for three sequences, in the case of a linear gap penalty. Given:

$$A = \begin{cases} a_1 = & (a_{1,1}, a_{1,2}, \ldots, a_{1,n_1}) \\ a_2 = & (a_{2,1}, a_{2,2}, \ldots, a_{2,n_2}) \\ a_3 = & (a_{3,1}, a_{3,2}, \ldots, a_{3,n_3}). \end{cases}$$

We proceed by computing the entries of a $(n_1+1)\times(n_2+1)\times(n_3+1)$-"'matrix"' $F(i,j,k)$. Actually F should be called a *multidimensional array*, or tensor.

After the computation, $F(n_1, n_2, n_3)$ will contain the best score α for a global alignment A^*. As in the pairwise case, we can use traceback to recover an actual alignment.

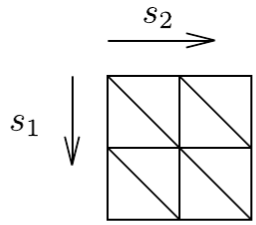
The main recursion is:

$$F(i,j,k) = \max \begin{cases} F(i-1,j-1,k-1) + s(a_{1i}, a_{2j}, a_{3k}), \\ \\ F(i-1,j-1,k) + s(a_{1i}, a_{2j}, -), \\ F(i-1,j,k-1) + s(a_{1i}, -, a_{3k}), \\ F(i,j-1,k-1) + s(-, a_{2j}, a_{3k}), \\ \\ F(i-1,j,k) + s(a_{1i}, -, -), \\ F(i,j-1,k) + s(-, a_{2j}, -), \\ F(i,j,k-1) + s(-, -, a_{3k}), \end{cases}$$

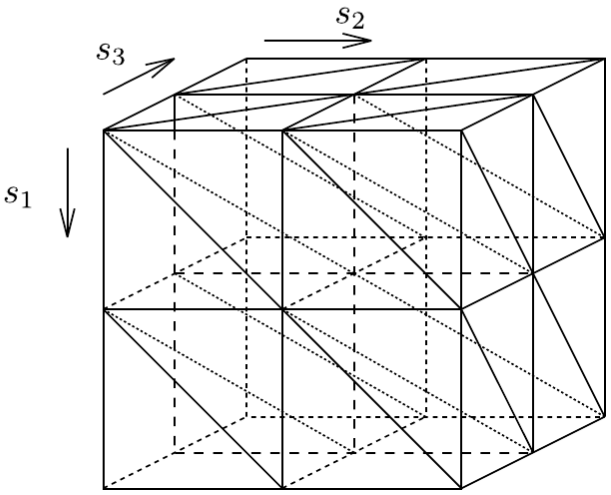
$$\text{for } 1 \le i \le n_1,\ 1 \le j \le n_2,\ 1 \le k \le n_3,$$

where $s(x, y, z)$ returns a score for a given column of symbols $x, y, z \in \Sigma \cup \{-\}$. We will discuss reasonable choices for the function s further below.

2D



3D



Clearly, this algorithm generalizes to r sequences.

However, the time complexity is $O(n_1 \cdot n_2 \cdot \ldots \cdot n_r)$ Hence, it is only practical for small numbers of short sequences, say $r = 5$ or 6.

2

The initialization of the boundary cells will be discussed in the exercises.

[2]Computing an MSA with optimal sum-of-pairs-score has been proved to be NP-complete (so, most likely, no polynomial time algorithm exists that solves this problem).

7.7 WSOP score

One of the most common scoring functions for MSA is the *(weighted) sum of pairs*, which is defined as the (weighted) sum of the score of all pairwise projections of the MSA, that is,

$$c(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_{i,j} \cdot c(\pi_{i,j}(A))$$

Each pair of sequences (i, j) can be given a different *weight* $w_{i,j}$. Note that $c(\pi_{i,j}(A))$ involves another summation over the columns of $\pi_{i,j}(A)$. However, if we assume linear gap costs, then we can rewrite this as follows:

$$c(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{h=1}^{l} w_{i,j} \cdot s(a_{i,h}, a_{j,h}) = \sum_{h=1}^{l} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_{i,j} \cdot s(a_{i,h}, a_{j,h})$$

The nice thing about WSOP is that we can move the "inner" summation (running over the column index h) "outside", to the front.

In the case $k = 3$, the DP algorithm mentioned above would use

$$s(x, y, z) = w_{1,2}s(x, y) + w_{1,3}s(x, z) + w_{2,3}s(y, z)\,.$$

Example.

Let $s(x, y) = \begin{cases} 3 & \text{for } x = y \text{ (match)} \\ -2 & \text{for } x \neq y, (x, y) \in \Sigma \times \Sigma \text{ (mismatch)} \\ 0 & \text{for } x = y = \text{'}-\text{'} \text{ (such columns are eliminated in projection)} \\ -1 & \text{otherwise (gaps)} \end{cases}$.

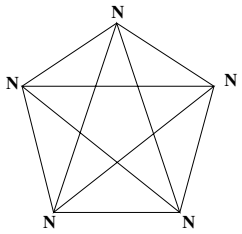
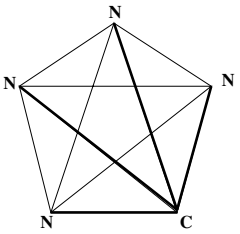
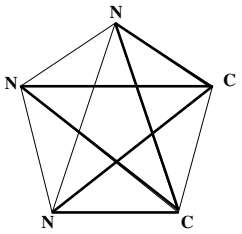
All the weight factors are 1.

```
          a1 = -  G  C  T  G  A  T  A  T  A  A  C  T
          a2 = G  G  G  T  G  A  T  -  T  A  G  C  T
          a3 = A  G  C  G  G  A  -  A  C  A  C  C  T
               ------------------------------------
score of column: -4  9 -1 -1  9  9  1  1 -1  9 -6  9  9 = 43
```

The sum of pair score takes all pairwise information into account; however it is easily biased if sequences from the same family are over-represented in the input. This disadvantage has to be dealt with by choosing the weights accordingly.

Multiple alignment:

		(1)		(2)		(3)	
s_1	...	N	...	N	...	N	...
s_2	...	N	...	N	...	N	...
s_3	...	N	...	N	...	N	...
s_4	...	N	...	N	...	C	...
s_5	...	N	...	C	...	C	...

Comparisons in each case:	(1)	(2)	(3)
			
# N-N pairs:	10	6	3
# N-C pairs:	0	4	6
# C-C pairs:	0	0	1
⇒ using BLOSUM50:	70	34	22

BLOSUM50 scores: N-N: 7, N-C: −2, C-C: 13.

An undesirable property of the WSOP cost function is the following:

Consider a position i in an SP-optimal multi-alignment A^* that is *constant*, i.e., has the same residue in all sequences.

What happens when we add a new sequence? If the number of aligned sequences is small, then we would not be too surprised if the new sequence shows a different residue at the previously constant position i.

However, if the number of sequences is large, then we would expect the constant position i to remain constant, if this is possible at all.

Unfortunately, the SP score favors the opposite behavior: the more sequences there are in an MSA, the easier it is (in relative terms), for a differing residue to be placed in an otherwise constant column!

Example:

$$\begin{array}{rccc} a_1 = & \ldots & \mathtt{E} & \ldots \\ \vdots & \ldots & \vdots & \ldots \\ a_{r-1} = & \ldots & \mathtt{E} & \ldots \\ a_r = & \ldots & \mathtt{E} & \ldots \end{array} \quad \leftrightarrow \quad \begin{array}{rccc} a_1 = & \ldots & \mathtt{E} & \ldots \\ \vdots & \ldots & \vdots & \ldots \\ a_{r-1} = & \ldots & \mathtt{E} & \ldots \\ a_r = & \ldots & \mathtt{C} & \ldots \end{array}$$

Using BLOSUM50, we obtain:

$$s(\mathtt{E}^r) = s(\mathtt{E},\mathtt{E}) \cdot \frac{r(r-1)}{2} = 6 \cdot \frac{r(r-1)}{2}.$$

The value for $s(\mathtt{E}^{r-1}\mathtt{C})$ is obtained from this by subtracting $(r-1) \cdot s(\mathtt{E},\mathtt{E})$ and then adding $(r-1) \cdot s(\mathtt{E},\mathtt{C})$. So, the difference between $s(\mathtt{E}^r)$ and $s(\mathtt{E}^{r-1},\mathtt{C})$ is:

$$(r-1)\Big(s(\mathtt{E},\mathtt{E}) - s(\mathtt{E},\mathtt{C})\Big) = (r-1)(6-(-3)) = 9(r-1).$$

Therefore, the relative difference is

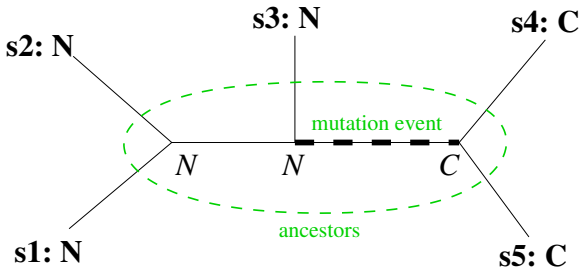
$$\frac{c(\mathtt{E}^r) - c(\mathtt{E}^{r-1}\mathtt{C})}{c(\mathtt{E}^r)} = \frac{9(r-1)}{6r(r-1)/2} = \frac{3}{r} \ ,$$

which *decreases* as the number of sequences r *increases!*

7.8 Scoring along a tree

Assume that we already have a *phylogenetic tree T* for the sequences that we want to align, i.e., a tree whose leaves are labeled by the sequences.

Then, instead of comparing *all pairs* of residues in a column of an MSA, one may instead determine an optimal labeling of the internal nodes of the tree by symbols in a given column (in this case col. (3) from the example) and then sum over *all edges in the tree*:

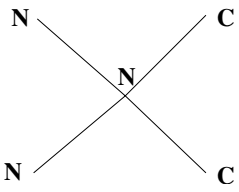


Such an optimal, i.e., *most parsimonious labeling* of the internal nodes can be computed in polynomial time using the *Fitch* algorithm.

Based on the present tree, the scores for columns (1), (2) and (3) are: $7 \times 7 = 49$, $6 \times 7 - 2 = 40$ and $4 \times 7 - 2 + 2 \times 13 = 52$.

7.9 Scoring along a star

In a third alternative, one sequence is treated as the ancestor of all others in a so-called *star phylogeny*:



Based on this star phylogeny, and assuming that sequence 1 is at the center of the star, the scores for columns (1), (2) and (3) are: $4 \times 7 = 28$, $3 \times 7 - 2 = 19$ and $2 \times 7 - 2 \times 2 = 10$.

7.10 Scoring Schemes

At present, there is no conclusive argument that gives any one scoring scheme more justification than the others. The sum-of-pairs score is widely used, but is problematic.

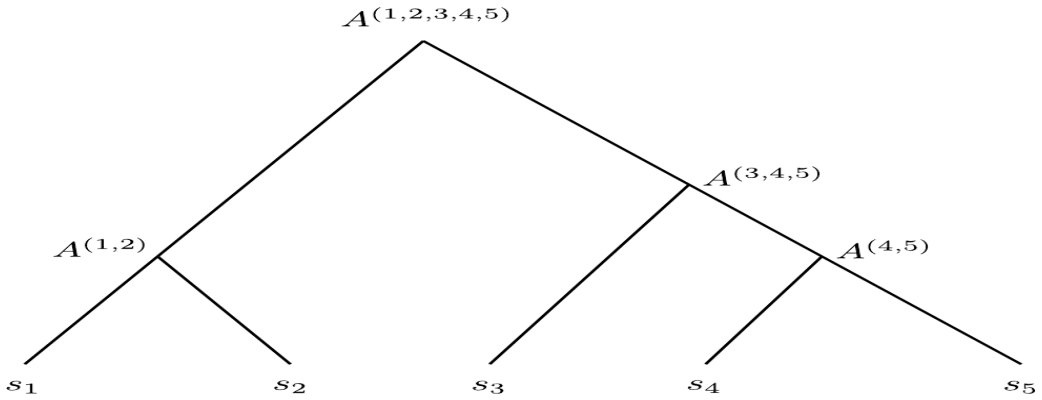
One advantage of the WSOP score (or cost) function is the ease with which gaps are modelled, since the score reduces to the summation of the pairwise scores, for which the handling of gaps is well understood.

7.11 Progressive alignment

The idea of progressive multiple sequence alignment is to compute a multiple alignment in a bottom-up fashion, starting with pairwise alignment and then combining them along a *guide tree*. The most popular multiple alignment programs follow this strategy.

In its simplest fashion, the given sequences a_i are added one after another to the growing multiple alignment, i.e. first a_1 and a_2 are aligned, then a_3 is added (we will see below how "adding" is actually done), then a_4, and so forth.

Ideally, the phylogenetic tree should be known and be used as the guide tree. In practice, we may have to use heuristic guide trees. Note that multiple alignments are often used to infer phylogenetic trees; both depend upon each other.



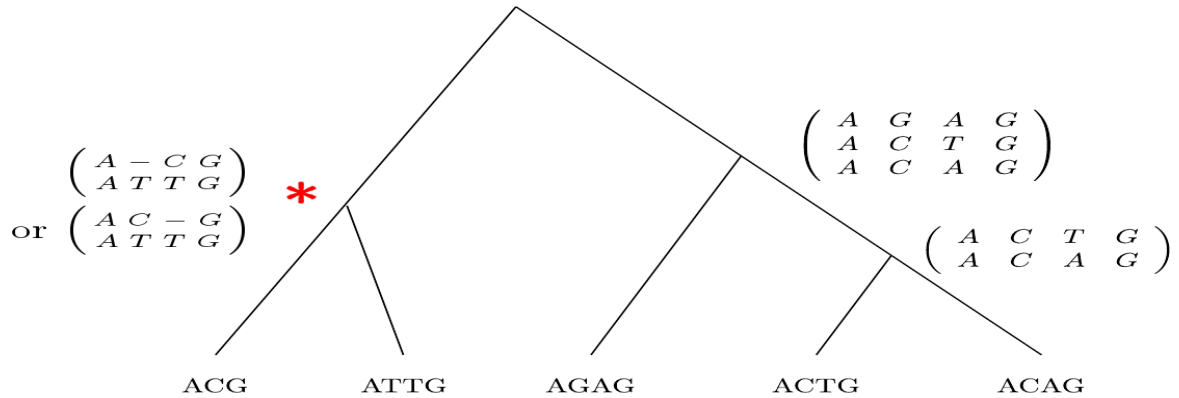
Progressive alignment along the branches of a phylogenetic tree. Sequences s_1 and s_2 are aligned, giving alignment $A^{(1,2)}$. Sequences s_4 and s_5 give $a^{(4,5)}$ which then is aligned with s_3 giving $A^{(3,4,5)}$. Finally, aligning $A^{(1,2)}$ and $A^{(3,4,5)}$ gives the multipe alignment of all sequences, $A^{(1,2,3,4,5)}$.

The main advantage of progressive methods is speed: Only pairwise alignments are computed.

A potential disadvantage is the "procedural" definiton: There is no well-founded global objective function being optimized. It may be difficult to evaluate the quality of the results.

The main disadvantage is of course the dependency on a guide tree.

The strict bottom-up progressive computation can also lead to problems. In the example below, it cannot be decided at the time of computing the alignment at the node marked with the asterisk (∗) which of the two alternatives is better. Only the rest of the tree indicates that the second one is probably the correct one.



7.12 Aligning alignments

An important subtask in progressive alignment is to align two existing multiple alignments. This is a *pairwise* alignment problem, but instead of simple letters we now have alignment columns to align.

We need the equivalent of a scoring matrix for alignment columns. One way to define such an extended score is to add the scores for all pairs of letters between the two columns.

For example the score for the two columns

$$A_{\cdot,i} = \begin{pmatrix} A \\ C \\ - \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} G \\ T \end{pmatrix} = B_{\cdot,j}$$

would be

$$s(A_{\cdot,i}, B_{\cdot,j}) := \sum_{\alpha} \sum_{\beta} s(A_{\alpha,i}, B_{\beta,j}) \,,$$

in the above example, $s(A,G) + s(A,T) + s(C,G) + s(C,T) + s(-,G) + s(-,T)$.

If the alignments are "deep" compared to the alphabet size, one can store the frequencies (= number of occurrences) of the letters in a table of size $|\Sigma|$ and replace the iteration over the rows of the columns by an iteration over the alphabet. This is also called *profile alignment*.

The following example illustrates the procedure for the alignment of

$$A = \begin{pmatrix} T & A & G \\ G & - & C \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} A & T & C & A & G \\ A & G & C & - & G \end{pmatrix}$$

using the unit cost model.

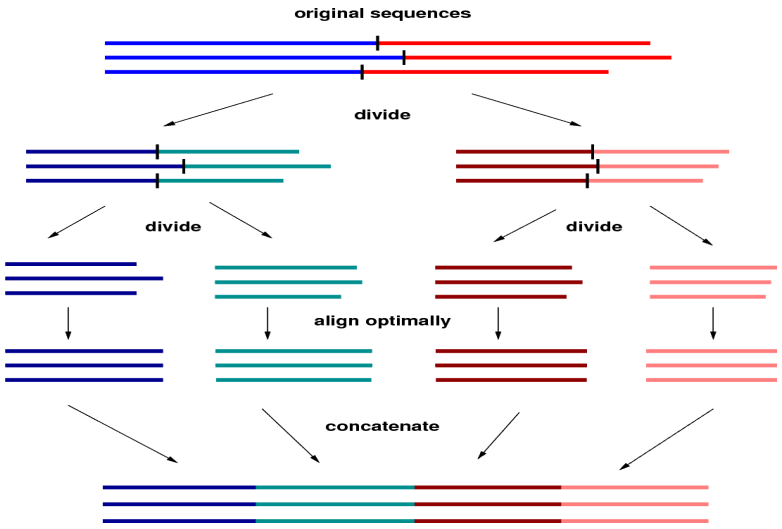
	$\begin{matrix} - \\ - \end{matrix}$	$\begin{matrix} A \\ A \end{matrix}$	$\begin{matrix} T \\ G \end{matrix}$	$\begin{matrix} C \\ C \end{matrix}$	$\begin{matrix} A \\ - \end{matrix}$	$\begin{matrix} G \\ G \end{matrix}$
$\begin{matrix} - \\ - \end{matrix}$	0	4	8	12	14	18
$\begin{matrix} T \\ G \end{matrix}$	4	4	6	10	12	16
$\begin{matrix} A \\ - \end{matrix}$	6	6	8	...		
$\begin{matrix} G \\ C \end{matrix}$	10					

7.13 Divide-and-conquer alignment

Another heuristic approach is to divide all sequences at suitable cut points into left and right parts, and then to solve the two subproblems separately using either recursion or, if the sequences are short enough, an exact algorithm.

Of course the main question is how to choose good cut positions. We will use the notation ++ for the concatenation of alignments.

Overview of divide-and-conquer multiple sequence alignment:



Algorithm 13.1 $DCA(s_1, s_2, \ldots, s_k; L)$

Let $n_p \leftarrow |s_p|$ for all p, $1 \leq p \leq k$
if $\max\{n_1, n_2, \ldots, n_k\} \leq L$ **then**
    return $MSA(s_1, s_2, \ldots, s_k)$
else
    $\hat{c}_1 \leftarrow \lceil n_1/2 \rceil$
    compute $(c_2, \ldots, c_k)$ such that $(\hat{c}_1, c_2, \ldots, c_k)$ is a C-optimal cut
    return $DCA(s_1[1..\hat{c}_1], s_2[1..c_2], \ldots, s_k[1..c_k]; L) +\!\!+ DCA(s_1[\hat{c}_1 + 1..n_1], s_2[c_2 + 1..n_2], \ldots, s_k[c_k + 1..n_k]; L)$

The cut positions are found heuristically using the so-called score-loss matrices, defined as follows:

Definition 11.2 Given two sequences s and t of lengths m and n, respectively, their **additional cost matrix** $C = (C(i,j))_{0 \leq i \leq m, 0 \leq j \leq n}$ is defined by

$$C(i,j) \quad = \quad \min \left\{ D(A +\!\!+ B) \;\middle|\; \begin{array}{l} A \text{ is an alignment of the prefixes} \\ \quad s[1 \ldots i] \text{ and } t[1 \ldots j] \\ B \text{ is an alignment of the suffixes} \\ \quad s[i+1 \ldots m] \text{ and } t[j+1 \ldots n] \end{array} \right\} - d(s,t)$$

where "++" denotes concatenation of alignments. The **score loss matrix** is defined similarly in terms of maximal score.

The idea is that $C(i, j)$ describes what we lose when the alignment is forced to pass through the (i, j) entry of the DP matrix.

Example (for additional cost):
The sequences are *CT* and *AGT*. We use unit cost. D is the DP matrix for global alignment. Likewise, D^{rev} is the DP matrix computed in reverse direction. Then the additional cost matrix C is optained by adding D and D^{rev} pointwise and subtracting the optimal score (here: 2) from each entry.

D :

	A	G	T	
C	0	1	2	3
T	1	1	2	3
	2	2	2	2

D^{rev} :

	A	G	T	
C	2	1	1	2
T	2	1	0	1
	3	2	1	0

C :

	A	G	T	
C	0	0	1	3
T	1	0	0	2
	3	2	1	0

Note that an optimal alignment uses the 0-entries of C only.

		C	T	A	T	A	C
	0	2	4	6	8	10	12
G	2	1	3	5	7	9	11
T	4	3	1	3	5	7	9
A	6	5	3	1	3	5	7
T	8	7	5	3	1	3	5
C	10	8	7	5	3	2	3

Figure 2a

	C	T	A	T	A	C	
G	3	4	3	4	6	8	10
T	4	2	3	2	4	6	8
A	6	4	2	2	2	4	6
T	8	6	4	2	1	2	4
C	10	8	6	4	2	0	2
	12	10	8	6	4	2	0

Figure 2b

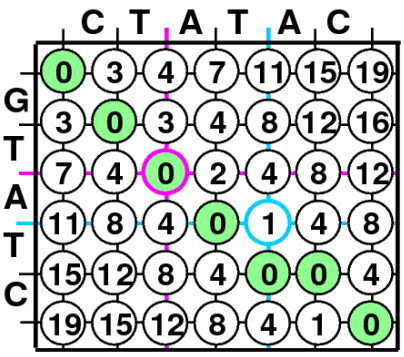


Figure 2c

Figures 2a and 2b show the standard dynamic programming distance matrices of the sequences **s** = GTATC and **t** = CTATAC (using unit cost and penalty +2 for each single insertion/deletion), computed from the upper left to the lower right (Fig. 2a) and from the lower right to the upper left (Fig. 2b). Figure 2c displays the *additional–cost* matrix, containing the values $C_{\mathbf{s},\mathbf{t}}[c,d]$ for each pair of slicing positions (c,d), e.g.

$$C_{\mathbf{s},\mathbf{t}}[2,2] = w_{opt}[\mathrm{CT},\mathrm{GT}] + w_{opt}[\mathrm{ATAC},\mathrm{ATC}] - w_{opt}[\mathrm{CTATAC},\mathrm{GTATC}] = 1 + 2 - 3 = 0$$

$$C_{\mathbf{s},\mathbf{t}}[4,3] = w_{opt}[\mathrm{CTAT},\mathrm{GTA}] + w_{opt}[\mathrm{AC},\mathrm{TC}] - w_{opt}[\mathrm{CTATAC},\mathrm{GTATC}] = 3 + 1 - 3 = 1$$

In each matrix, the optimal alignment path is colored green. Its additional–cost matrix entries are, of course, zero.

The **multiple additional cost** of a family of cut positions $(c_1, c_2, \ldots, c_k)$ is defined as

$$C(c_1, c_2, \ldots, c_k) := \sum_{1 \le p < q \le k} C_{(p,q)}(c_p, c_q),$$

where $C_{(p,q)}$ is the additional cost matrix for the pair (p, q).

The DQA tries to minimize the mulitple additional cost when choosing the cut positions. However, simple examples exist showing that an optimal cut need not have multiple additional cost zero, and cut with smallest possible multiple additional cost need not be an optimal cut.

5 Multiple Match Refinement and T-Coffee

In this section we describe how to extend the match refinement to the multiple case and then use *T-Coffee* to heuristically compute a multiple trace.

This exposition is based on the following sources, which are all recommended reading:

1. Notredame, Higgins, Heringa: T-Coffee, a Novel Method for Fast and Accurate Multiple Sequence Alignment, Journal of Molecular Biology, 2000, Vol 302, pages 205-217.
2. Rausch, Emde, Weese, Döring, Notredame, Reinert: Segment-based multiple sequence alignment, ECCB 2008, Cagliari

5 Multiple Match refinement

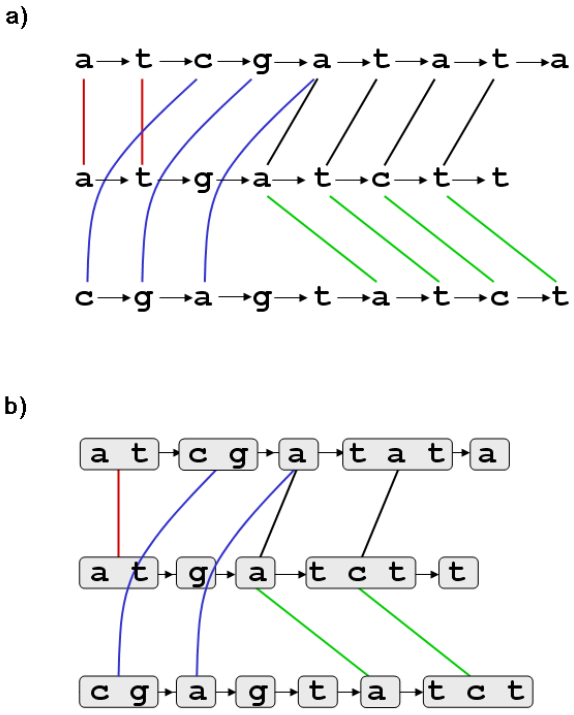
We start by extending the pairwise match refinement to the multiple case. The result of the multiple match refinement naturally induces an input graph for the *multiple trace problem* which is NP-hard but could be solved with ILP based techniques.

However, in practice we can resort to more efficient, but heuristic methods. In the second part of the lecture we will describe the TCoffee algorithm which is basically a heuristic for the multiple trace problem (although originally not advertised as such).

5.1 Multiple Match refinement

Lets start by extending the definitons from the pairwise case to the multiple case.

Definition 1. Let $\mathcal{S} = \{S^1, S^2, ..., S^n\}$ be a set of n sequences with $S^p = s^p_1 s^p_2, ..., s^p_{|S^p|}$, $p \in \{1, 2, ..., n\}$. A *segment match* $M = (S^p_{ij}, S^q_{kl})$ is an alignment between the two *segments* $S^p_{ij} = s^p_{i+1} s^p_{i+2} ... s^p_j$ and $S^q_{kl} = s^q_{k+1} s^q_{k+2} ... s^q_l$ with $p, q \in \{1, 2, ..., n\}$ and $p \neq q$ and $0 \leq i \leq j \leq |S^p|$ and $0 \leq k \leq l \leq |S^q|$.



5.2 T-Coffee

We now discuss a simple multiple alignment algorithm that outperforms ClustalW on certain data sets, namely T-Coffee. T-Coffee stands for **T**ree based **C**onsistency **O**bjective **F**unction **F**or alignm**E**nt **E**valuation. Its basic idea consists of combining global and local sequence information.

It uses the same progressive alignment method as ClustalW but tries to rectify the greedy choices made by incorporating information from *all* pairs of sequences.

The basic steps of T-Coffee are:

1. Generate *primary* libraries of alignments.
2. Derive library *weights*.
3. *Combine* libraries into single primary library.
4. *Extend* the library.
5. Use the extended library for progressive alignment.

5.3 Generating primary libraries

A primary library in T-Coffee contains a set of pairwise alignments and could be derived using any method. By default two libraries are generated:

1. A library of all global pairwise alignments using ClustalW.

2. A library of the ten top-scoring local alignments using Lalign from the FASTA package.

All these alignments contain information that is more or less reliable. Hence T-Coffee combines them to produce more reliable alignments.

5.4 Derive library weights

Each library is weighted with the percent identity, a measure which is known to be a reasonable indicator when aligning sequences with more than 30% identity.

We give now an example for computing a global primary library and weighting it. Assume we are given the four strings:

```
s1 = GARFIELD THE LAST FAT CAT
s2 = GARFIELD THE FAST CAT
s3 = GARFIELD THE VERY FAST CAT
s4 = THE FAT CAT
```

The regular progressive alignment method would produce the following alignment:

```
a1 = GARFIELD THE LAST FA-T CAT
a2 = GARFIELD THE FAST CA-T
a3 = GARFIELD THE VERY FAST CAT
a4 = -------- THE ---- FA-T CAT
```

Obviously the second `CAT` is not correctly aligned. T-Coffee would first produce the following global, primary library.

The percent identity is computed on *matching* positions. (Ignore the blanks, they are inserted for better readibility.)

```
a1=GARFIELD THE LAST FAT CAT        a2=GARFIELD THE ---- FAST CAT
a2=GARFIELD THE FAST CAT ---        a3=GARFIELD THE VERY FAST CAT
     weight=88                           weight=100

a1=GARFIELD THE LAST FA-T CAT       a2=GARFIELD THE FAST CAT
a3=GARFIELD THE VERY FAST CAT       a4=-------- THE FA-T CAT
     weight=80                           weight=100

a1=GARFIELD THE LAST FAT CAT        a3=GARFIELD THE VERY FAST CAT
a4=-------- THE ---- FAT CAT        a4=-------- THE ---- FA-T CAT
     weight=100                          weight=100
```

5.5 Combine libraries

If there is more than one primary library available, they are combined by merging any pair that is duplicated between the two libraries. The combined pair has as a new weight the sum of the two individual weights.

After combining the primary libraries into a single primary library, this library is extended by incorporating consistency information.

5.6 Extending the primary library

Finding the highest weighted, consistent subsets of pairwise contraints is known as the *maximum weight trace* problem and is by itself NP-hard.

Hence T-Coffee employs a heuristic strategy with the goal to compute weights that reflect the information contained in the whole library. To do so it employs a *triplet* approach. This works as follows for each pair of sequences s_i, s_j.

1. Align s_i and s_j *through* another sequence s_l, $1 \leq l \neq i, j \leq k$. For each column c of the alignment $A(s_i, s_j)$ that contains no gap character, we set the initial weight of the match $A[i, c] \leftrightarrow A[j, c]$ to the primary library's weight of the alignment of the x-th character of s_i with the y-th character of s_j. The weight of columns containing gap characters is set to 0.
2. Take the minimum weight of the primary library alignments between s_i, s_l and s_l, s_j if the x-th character of s_i is aligned to the z-th character of s_l and the z-th character of s_l is aligned to the y-th of s_j. The weight is added to the inital weight of the match $x \leftrightarrow y$.

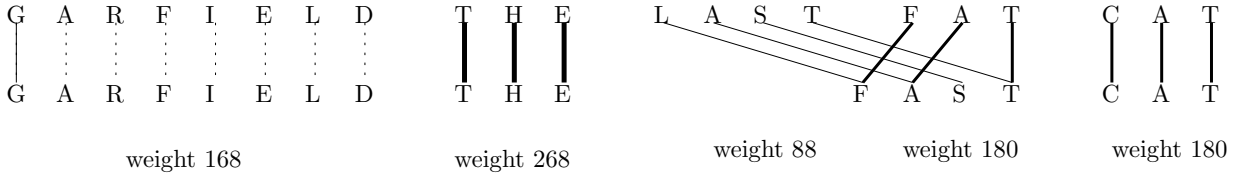
Lets go back to our example. We want to extend the primary library for the pair s_1, s_2. Note that the primary library has uniform weights since we did not combine more than one.

```
a1=GARFIELD THE LAST FAT CAT
   ******** ***  ***  **       weight = 88
a2=GARFIELD THE FAST CAT ---

a1=GARFIELD THE LAST FA-T CAT
   ******** ***      ** * ***
a3=GARFIELD THE VERY FAST CAT  weight = 80
   ******** ***      **** ***
a2=GARFIELD THE ---- FAST CAT

a1=GARFIELD THE LAST FA-T CAT
            ***      ** * ***
a4=         THE      FA-T CAT  weight = 100
            ***      ** * ***
a2=GARFIELD THE      FAST CAT
```

We combine the above extensions into the weighting scheme for the pair of sequences s_1, s_2 (spot the error in the figure !).



Finally we use the above weighting scheme to compute the pairwise alignment using a dynamic programming algorithm without gap penalties. All the information is already incorporated into the libraries. In our case the best pairwise alignment would be:

```
a1=GARFIELD THE LAST FA-T CAT
a2=GARFIELD THE ---- FAST CAT
```

This will correct the error in the inital alignment.

5.7 Computing the progressive alignment

In order to compute the progressive alignment we compute the distance matrix using the extended alignment library as computed above. This again is used to compute a guide tree using the neighbor joining method.

The gaps introduced in the first alignment are fixed and cannot be shifted later.

When aligning two groups of sequences (containing possibly only one sequence) the *average* score from the extended libraries is used for each column.

5.8 Combining TCoffee with the multiple srm

The just presented, original T-Coffee uses *libraries* which are simply alignment graphs where each vertex corresponds to a single residue or nucleotide. Similar to T-Coffee's library, the alignment graph contains alignment information about a set of sequences $\mathcal{S} = \{S^0, S^1, ..., S^{n-1}\}$.

Given such an initial alignment graph, we apply the triplet extension introduced in T-Coffee. Whereas T-Coffee ensures consistency on pairs of characters we ensure consistency on pairs of vertices.

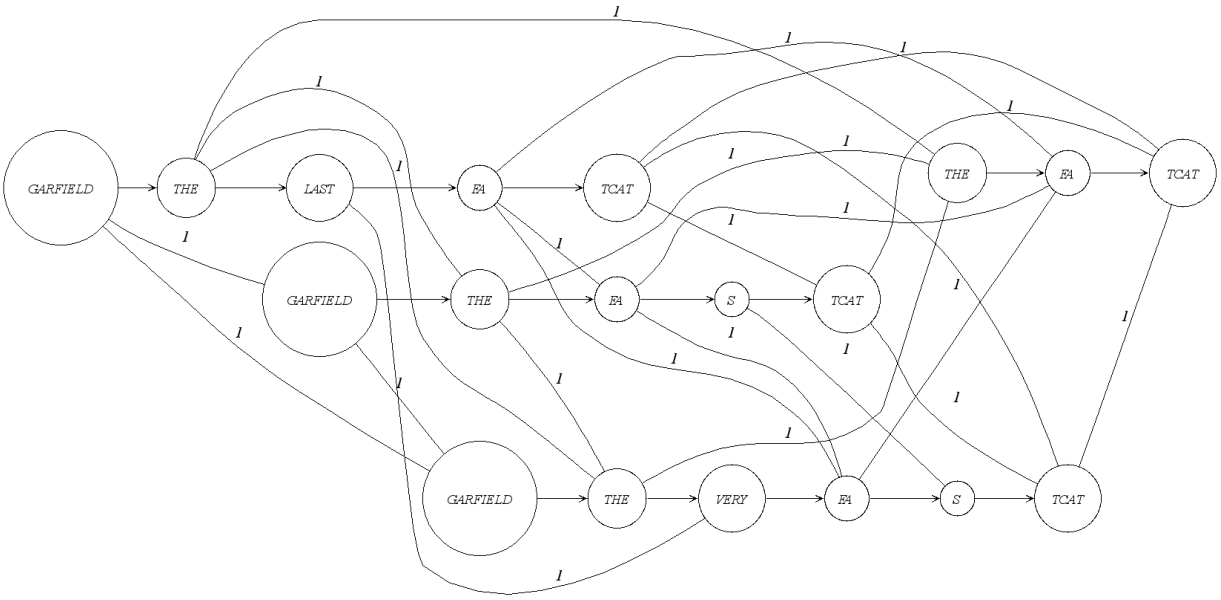
5.9 Combining TCoffee with the multiple srm

Similarly, a progressive alignment is possible on graphs, i.e., a pairwise alignment simply aligns two strings of vertices instead of sequence characters.

A profile in terms of an alignment graph is a string where each position has a set of vertices. Thus, given a guide tree we can perform a progressive alignment on an alignment graph of multiple sequences in the same manner as aligning the sequences themselves.

The approach is, however, more generic because a single vertex can be any group of characters, e.g., a large segment, a gene, or just a single character.

5.10 Combining TCoffee with the multiple srm



5.11 Combining TCoffee with the multiple srm

The advantage of our method is that the input can be *any* set of segment matches. To illustrate it, we explain two different mechanisms to generate segment matches.

For protein alignments and short DNA sequences the standard match generation algorithm takes a set of sequences and builds all pairwise global and local alignments using the dynamic programming algorithms of Gotoh or Waterman and Eggert. We compute the dynamic programming matrix column-wise and save the traceback pointers in a reduced alphabet that can be efficiently stored. This enables us to use these algorithms for fairly long sequences, e.g., a set of adenovirus genomes.

However, for very long (genomic) sequences algorithms using quadratic space are impossible to use and space-efficient dynamic programming algorithms become too inefficient. For these problem instances we either use the enhanced suffix arrays to compute maximal unique matches or external tools such as BLAST.

5.12 Combining TCoffee with the multiple srm

Aligner	= 6	≥ 5	≥ 4	≥ 3	Avg. identity	CPU Time (s)
DIALIGN-T	7888	12161	18187	27690	48%	1259
SeqAn::T-Coffee	12795	18525	25147	32396	63%	1751
MAFFT*	12450	18011	24624	32084	62%	118
MUSCLE*	50	817	5257	21849	38%	673
SeqAn::T-Coffee*	**12911**	**20078**	**27011**	**33147**	**65**%	328

Alignment of 6 adenoviruses: Running time and alignment quality of an alignment of 6 adenoviruses. The number of columns with at least 6, 5, 4, and 3 identical characters are reported together with the average identity.

As it can be seen the combined match refinement and TCoffee approach outperforms the currently best programs in the field.

5.13 Summary

You should know:

- Pairwise and multiple match refinement are efficient methods to preprocess overlapping segment matches.
- In the worst case, segment match refinement can refine a set of input matches to single base matches.
- TCoffee is a consistency based alignment algorithm that computes heuristically a multiple trace.
- The combination of multiple match refinement and TCoffee results in a versatile method to compare sequences based on a set of input segment matches.