1. Multiple Sequence Alignment

Slides for the Lecture on
Foundations of Sequence Analysis
Winter 2022/2023
Stefan Kurtz

January 17, 2023

- Up until now we always have considered comparing two sequences.
- This section is devoted to the comparison of $k \geq 2$ sequences to obtain multiple alignments.
- For example, consider the following protein sequences (small sections of six tyrosine kinase sequences):

```
>SRC_RSVP
FPIKWTAPEAALYGRFTIKSDVWSFGILLTELTTKGRVPYPGMVNREVLDQVERG
>YES_AVISY
FPIKWTAPEAALYGRFTIKSDVWSFGILLTELVTKGRVPYPGMVNREVLEQVERG
>ABL_MLVAB
FPIKWTAPESLAYNKFSIKSDVWAFGVLLWEIATYGMSPYPGIDLSQVYELLEKD
>FES_FSVGA
QVPVKWTAPEALNYGRYSSESDVWSFGILLWETFSLGASPYPNLSNQQTREFVEKG
>FPS_FUJSV
QIPVKWTAPEALNYGWYSSESDVWSFGILLWEAFSLGAVPYANLSNQQTREAIEQG
>KRAF_MSV36
TGSVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMAGELPYAHINNRDQIIFMVGRG
```

We would like to obtain a multiple alignment like this:

```
-FPIKWTAPEAALY---GRFTIKSDVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVERG
-FPIKWTAPEAALY---GRFTIKSDVWSFGILLTELVTKGRVPYPGMVNR-EVLEQVERG
-FPIKWTAPESLAY---NKFSIKSDVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLEKD
QVPVKWTAPEALNY---GRYSSESDVWSFGILLWETFSLGASPYPNLSNQ-QTREFVEKG
QIPVKWTAPEALNY---GWYSSESDVWSFGILLWEAFSLGAVPYANLSNQ-QTREAIEQG
TGSVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRDQIIFMVGRG
```

- Multiple sequence alignments are usually constructed for a set of sequences that are assumed to be homologous (i.e., they have a common ancestor sequence) and the goal is usually to detect homologous residues or bases and place them in the same column of the multiple alignment.
- Of course, we want to arrange the symbols in the columns in an optimal way to reveal as much as possible differences and similarities of the sequences.
- Therefore, we need to score multiple alignments.
- In the following, we will formally define the notion of multiple alignment and describe two approaches for scoring multiple alignments.
- Moreover, we present several methods to compute multiple alignments with optimal or near optimal score.

- Suppose that we are given a set S of k sequences $s_1, \dots, s_k$ over some alphabet $\mathcal{A}$.
- Let $n_i = |s_i|$ for each i, $1 \leq i \leq k$.
- Let $- \notin \mathcal{A}$ be a gap-symbol, and $\mathcal{A}' = \mathcal{A} \cup \{-\}$.
- To get rid of gaps in sequences we define a function $\mathit{delgap} : (\mathcal{A}')^* \to \mathcal{A}^*$ by

$$\begin{aligned}
\mathit{delgap}(a) &= a \text{ for each } a \in \mathcal{A} \\
\mathit{delgap}(-) &= \varepsilon \\
\mathit{delgap}(w[1 \dots n]) &= \mathit{delgap}(w[1]) \dots \mathit{delgap}(w[n]) \\
&\quad \text{for each sequence } w \in (\mathcal{A}')^n
\end{aligned}$$

Definition 1

A *multiple sequence alignment* (*MSA*, for short) of S is a sequence of k strings $(s'_1, \ldots, s'_k)$ satisfying the following properties:

1. There is some $\ell \geq 1$ such that $s'_i \in (\mathcal{A}')^\ell$ for all i, $1 \leq i \leq k$,
2. $delgap(s'_i) = s_i$ for all i, $1 \leq i \leq k$, i.e. deleting all gap symbols in s'_i gives s_i
3. for all j, $1 \leq j \leq \ell$, there is at least one i, $1 \leq i \leq k$ such that $s'_i[j] \neq -$, i.e. there is no column in the *MSA* consisting of gaps only.

$\ell = |s'_k|$ is the length of the *MSA* $(s'_1, \ldots, s'_k)$ and it holds:

$$\max\{n_i \mid 1 \leq i \leq k\} \leq \ell \leq \sum_{i=1}^{k} n_i$$

As with pairwise alignments we display an *MSA* by placing the s'_i on top of each other, see the example at the beginning of this section.

Scoring MSAs by consensus costs

- We consider two basic approaches to assign a quantitative measure to *MSA*s, i.e. a cost or a score.
- The first approach is based on the notion of consensus.
- The consensus of a column is a non-gap symbol that occurs most often in the column.
- The consensus sequence is the sequence of consensi of the columns.
- Counting in all columns the number of symbols which are not identical to the consensus symbol of the column leads to a cost notion.
- Consider an *MSA* $(s'_1, \ldots, s'_k)$ of the sequences $s_1, \ldots, s_k$.
- Let ℓ be the length of the *MSA*.
- For each j, $1 \leq j \leq \ell$ and $a \in \mathcal{A}$ define $count(j, a) = |\{i \mid 1 \leq i \leq k, s'_i[j] = a\}|$, i.e. $count(j, a)$ is the number of occurrences of character a in the jth column of the *MSA*.

Scoring MSAs by consensus costs

- A sequence *cons* of length ℓ is a *consensus* of $(s'_1, \ldots, s'_k)$ if for each j, $1 \leq j \leq \ell$, $count(j, cons[j]) \geq count(j, a)$ for all characters $a \in \mathcal{A}$.
- That is, a consensus sequence consists of characters occurring most often in the corresponding column of the MSA.

$$dist(cons, (s'_1, \ldots, s'_k)) = \sum_{j=1}^{\ell} (k - count(j, cons[j]))$$

is the *distance* of the consensus *cons* and the *MSA*.

Example 1

Let $s_1 = \texttt{AATGCT}$, $s_2 = \texttt{ATTC}$, $s_3 = \texttt{TCC}$. An *MSA* of these sequences is as follows, with a consensus shown below the bottom line:

$$\begin{array}{l|cccccc}
s'_1 & \texttt{A} & \texttt{A} & \texttt{T} & \texttt{G} & \texttt{C} & \texttt{T} \\
s'_2 & \texttt{A} & - & \texttt{T} & \texttt{T} & \texttt{C} & - \\
s'_3 & - & - & - & \texttt{T} & \texttt{C} & \texttt{C} \\
\hline
\textit{cons} & \texttt{A} & \texttt{A} & \texttt{T} & \texttt{T} & \texttt{C} & \texttt{T}
\end{array}$$

We have $\mathit{dist}(\mathit{cons}, (s'_1, s'_2, s'_3)) = 1 + 2 + 1 + 1 + 0 + 2 = 7$.

It does not matter which of the possible consensus sequences we take, the distance to the given *MSA* is invariant:

Lemma 1

Let $cons_1$ *and* $cons_2$ *be two different consensi of the same MSA* $(s'_1, \ldots, s'_k)$. *Then* $dist(cons_1, (s'_1, \ldots, s'_k)) = dist(cons_2, (s'_1, \ldots, s'_k))$.

Proof.

Lets consider a particular column of the MSA, i.e. lets fix j, $1 \leq j \leq \ell$. *First note that neither*

$$\begin{aligned} &count(j, cons_1[j]) > count(j, cons_2[j]) \textit{ nor} \\ &count(j, cons_1[j]) < count(j, cons_2[j]). \end{aligned}$$

Hence $count(j, cons_1[j]) = count(j, cons_2[j])$, *i.e.* $cons_1[j]$ *occurs equally often as* $cons_2[j]$. *As a consequence*

$$k - count(j, cons_1[j]) = k - count(j, cons_2[j])$$

Since this holds for each j, $1 \leq j \leq \ell$, *the sum of the sizes over all* j *and thus the distance is identical.* $\square$

Scoring MSAs by consensus costs

- Since the distance is not influenced by the choice of the consensus, it makes sense to evaluate an *MSA* according to the distance to a consensus.
- That is, we define the consensus costs of an MSA $(s'_1, \ldots, s'_k)$ by

$$conscost(s'_1, \ldots, s'_k) = dist(cons, (s'_1, \ldots, s'_k))$$

  where *cons* is a consensus of $(s'_1, \ldots, s'_k)$.
- The smaller the cost, the better the multiple alignment.
- If most columns contain a dominating character, the consensus cost is small.
- If the characters in the columns vary, the consensus cost is large.

Scoring MSAs by consensus costs

– The *MSA*-problem with consensus costs consists of of finding the *MSA* $(s'_1, \ldots, s'_k)$ of the given sequences $s_1, \ldots, s_k$ such that the consensus cost $conscost(s'_1, \ldots, s'_k)$ is minimal.

Scoring MSAs by sums of pairwise scores

- The second optimization criterion for *MSA*s is based on pairwise distances.
- Let δ be a function satisfying $\delta(\alpha \rightarrow \beta) \geq 0$ for each edit operation $\alpha \rightarrow \beta$; extend it by requiring $\delta(- \rightarrow -) = 0$.
- The score is determined by computing the sum of scores (according to δ) for all pairs of the characters and gaps in each column of the *MSA*.
- Hence it is called *sum-of-pairs scoring* (SP-scoring, for short).
- Consider an *MSA* $(s'_1, \ldots, s'_k)$ of length ℓ.

For each j, $1 \leq j \leq \ell$ we define the SP-column score

$$\delta_{\mathsf{SP}}(s'_1[j], \ldots, s'_k[j]) = \sum_{i=1}^{k-1} \sum_{r=i+1}^{k} \delta(s'_i[j] \rightarrow s'_r[j])$$

The SP-alignment score is then defined as the sum of all SP-column scores:

$$\delta_{\mathsf{SP}}(s'_1, \ldots, s'_k) = \sum_{j=1}^{\ell} \delta_{\mathsf{SP}}(s'_1[j], \ldots, s'_k[j])$$

Example 2

Let $s_1 = \texttt{AATGCT}$, $s_2 = \texttt{ATTC}$, $s_3 = \texttt{TCC}$ be the sequences from Example 1. Consider the following *MSA* of s_1, s_2, s_3, that is $k = 3$ and $\ell = 6$:

s_1'	A	A	T	G	C	T
s_2'	A	-	T	T	C	-
s_3'	-	-	-	T	C	C

For all $\alpha, \beta \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}, -\}$ define $\delta(\alpha \to \beta) = \begin{cases} 0 & \text{if } \alpha = \beta \\ 1 & \text{otherwise} \end{cases}$

$$\begin{aligned}
\delta_{\mathsf{SP}}(s_1', s_2', s_3') &= \sum_{j=1}^{6} \sum_{i=1}^{2} \sum_{r=i+1}^{3} \delta(s_i'[j], s_r'[j]) \\
&= \sum_{j=1}^{6} (\delta(s_1'[j], s_2'[j]) + \delta(s_1'[j], s_3'[j]) + \delta(s_2'[j], s_3'[j])) \\
&= (0+1+1) + (1+1+0) + (0+1+1) + \\
&\quad (1+1+0) + (0+0+0) + (1+1+1) \\
&\quad = 2+2+2+2+0+3 = 11.
\end{aligned}$$

Scoring MSAs by sums of pairwise scores

- The *MSA*-problem with SP-scoring is to find an *MSA* of the given sequences with minimum SP-alignment score.
- The minimum score is denoted by $\delta_{\text{opt}-\text{SP}}(s_1, \ldots, s_k)$.
- Note that for both scoring models we consider here, minimization is the optimization criterion.
- Despite this fact, we often use the notion "similarity", which is usually used in combination with maximizing scores.

Computing optimal multiple sequence alignments

The dynamic programming techniques for pairwise sequence alignments employs the following recurrence:

$$E_\delta(i,j) = \begin{cases} 0 & \text{if } i=0 \text{ and } j=0 \\ E_\delta(0,j-1)+\delta(\varepsilon \to v[j]) & \text{if } i=0 \text{ and } j>0 \\ E_\delta(i-1,0)+\delta(u[i] \to \varepsilon) & \text{if } i>0 \text{ and } j=0 \\ \min \left\{ \begin{array}{l} E_\delta(i-1,j)+\delta(u[i] \to \varepsilon) \\ E_\delta(i,j-1)+\delta(\varepsilon \to v[j]) \\ E_\delta(i-1,j-1)+\delta(u[i] \to v[j]) \end{array} \right\} & \text{if } i>0 \text{ and } j>0 \end{cases}$$

- This can be generalized to *MSA*s in a straightforward way: One computes an $(n_1+1) \times \cdots \times (n_k+1)$-table M such that for each $(q_1, \ldots, q_k) \in [0, n_1] \times \cdots \times [0, n_k]$, $M(q_1, \ldots, q_k)$ is the score of the optimal *MSA* for the prefixes $s_1[1 \ldots q_1], \ldots, s_k[1 \ldots q_k]$.
- Next we develop a recurrence for M.
- In analogy to the pairwise case, let q_1, ..., q_k be arbitrary but fixed (in the range specified above) and consider all possible last columns of an *MSA* of $s_1[1 \ldots q_1], \ldots, s_k[1 \ldots q_k]$.

Example 3

Consider the 4 sequences $s_1 =$ CATT, $s_2 =$ TATC, $s_3 =$ TCA and $s_4 =$ ATG. Here are the 15 possible last columns of any multiple alignment of $S = \{s_1, \ldots, s_4\}$:

T	T	T	T	T	T	T	T	-	-	-	-	-	-	-
C	C	C	C	-	-	-	-	C	C	C	C	-	-	-
A	A	-	-	A	A	-	-	A	A	-	-	A	A	-
G	-	G	-	G	-	G	-	G	-	G	-	G	-	G

That is, each last column is a combination of the gap symbol or the last character of the corresponding sequence:

- T for row 1,
- C for row 2,
- A for row 3,
- G for row 4.

Note that there is no column consisting of gap-symbols only.

In general, a last column of the MSA consists of any combination of gaps and characters $s_i[q_i]$, $1 \leq i \leq k$. A column of gap-symbols only is not allowed. To describe these combinations one enumerates all k-tuples

$$(d_1, \dots, d_k) \in \{0,1\} \times \dots \times \{0,1\} \setminus \{(0,\dots,0)\}.$$

$d_i = 0$ means that the ith character in the last column is the gap-symbol.
$d_i = 1$ means that the ith character in the last column is $s_i[q_i]$.

Example 4

For $s_1 =$ `CATT`, $s_2 =$ `TATC`, $s_3 =$ `TCA`, $s_4 =$ `ATG` the 15 last columns

T	T	T	T	T	T	T	T	-	-	-	-	-	-	-
C	C	C	C	-	-	-	-	C	C	C	C	-	-	-
A	A	-	-	A	A	-	-	A	A	-	-	A	A	-
G	-	G	-	G	-	G	-	G	-	G	-	G	-	G

of any multiple alignment of $S = \{s_1, \dots, s_4\}$ can be represented by:

1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Computing optimal multiple sequence alignments

- If the ith row of the last column is character $s_i[q_i]$ (case $d_i = 1$), then $s_i[q_i]$ is aligned.
- So in dimension i, one has to align the prefix $s_i[1 \dots q_i - 1] = s_i[1 \dots q_i - d_i]$, i.e. one has to refer to the entries in $M(\dots, q_i - 1, \dots) = M(\dots, q_i - d_i, \dots)$.
- To obtain the character which appears in the last column, define a function φ by $\varphi(q_i, 1) = s_i[q_i]$.
- If the ith row of the last column is a gap (case $d_i = 0$), then $s_i[q_i] = s_i[q_i - d_i]$ is not aligned.
- So in dimension i, one has to align the prefix $s_i[1 \dots q_i] = s_i[1 \dots q_i - d_i]$, i.e. one has to refer to the entries in $M(\dots, q_i - 0, \dots) = M(\dots, q_i - d_i, \dots)$.
- To obtain the gap appearing in the last column define $\varphi(q_i, 0) = -$.
- So in both cases, one has to refer to the entry $M(\dots, q_i - d_i, \dots)$ and $\varphi(q_i, d_i)$ appears in row i of the last column of the MSA.
- Assuming that the minimum of an empty set is 0, we thus obtain the following recurrence for M:

Computing optimal multiple sequence alignments

$$
\begin{aligned}
&M(q_1, \ldots, q_k) \\
&= \min\{\overbrace{M(q_1 - d_1, \ldots, q_k - d_k)}^{\text{previous entry}} + \\
&\qquad \textit{colcost}(\underbrace{\varphi(q_1, d_1), \ldots, \varphi(q_k, d_k)}_{\text{last column}}) \mid (d_1, \ldots, d_k) \in \{0,1\} \times \cdots \times \{0,1\}, \\
&\qquad\qquad (d_1, \ldots, d_k) \neq (0, \ldots, 0), \\
&\qquad\qquad q_1 \geq d_1, \ldots, q_k \geq d_k\}
\end{aligned}
$$

where

$$
\varphi(q_i, d_i) = \begin{cases} s_i[q_i] & \text{if } d_i = 1 \\ - & \text{otherwise} \end{cases}
$$

and *colcost* is a function assigning costs to a column of k elements from $\mathcal{A}'$, depending on the chosen model.

– If the model is based on the consensus cost, then $\textit{colcost} = \textit{conscost}$.

– In the SP-scoring model, $\textit{colcost} = \delta_{\mathsf{SP}}$.

Example 5

To exemplify the equation, let us partially evaluate it for $k = 2$. In the initial simplification step we obtain the following:

$$M(q_1, q_2) = \min\{M(q_1 - d_1, q_2 - d_2) + \mathit{colcost}(\varphi(q_1, d_1), \varphi(q_2, d_2)) \mid \begin{array}{l}(d_1, d_2) \in \{0,1\} \times \{0,1\},\\ (d_1, d_2) \neq (0,0),\\ q_1 \geq d_1, q_2 \geq d_2\}\end{array}$$

Now assume that $q_1 \geq 1$ and $q_2 \geq 1$. As $d_1 \leq 1$ and $d_2 \leq 1$, we can conclude $q_1 \geq d_1$ and $q_2 \geq d_2$. Then the right hand side simplifies to:

$$\begin{aligned}
&= \min \{M(q_1 - d_1, q_2 - d_2) + \mathit{colcost}(\varphi(q_1, d_1), \varphi(q_2, d_2)) \mid (d_1, d_2) \in \{(1,0), (0,1), (1,1)\}\} \\
&= \min \{M(q_1 - d_1, q_2 - d_2) + \mathit{colcost}(\varphi(q_1, d_1), \varphi(q_2, d_2)) \mid (d_1, d_2) = (1,0)\} \cup \\
&\qquad \{M(q_1 - d_1, q_2 - d_2) + \mathit{colcost}(\varphi(q_1, d_1), \varphi(q_2, d_2)) \mid (d_1, d_2) = (0,1)\} \cup \\
&\qquad \{M(q_1 - d_1, q_2 - d_2) + \mathit{colcost}(\varphi(q_1, d_1), \varphi(q_2, d_2)) \mid (d_1, d_2) = (1,1)\} \\
&= \min \{M(q_1 - 1, q_2 - 0) + \mathit{colcost}(\varphi(q_1, 1), \varphi(q_2, 0))\} \cup \\
&\qquad \{M(q_1 - 0, q_2 - 1) + \mathit{colcost}(\varphi(q_1, 0), \varphi(q_2, 1))\} \cup \\
&\qquad \{M(q_1 - 1, q_2 - 1) + \mathit{colcost}(\varphi(q_1, 1), \varphi(q_2, 1))\} \\
&= \min \{M(q_1 - 1, q_2) + \mathit{colcost}(s_1[q_1], -)\} \cup \\
&\qquad \{M(q_1, q_2 - 1) + \mathit{colcost}(-, s_2[q_2])\} \cup \\
&\qquad \{M(q_1 - 1, q_2 - 1) + \mathit{colcost}(s_1[q_1], s_2[q_2])\}
\end{aligned}$$

Example 5

Now assume $q_1 \geq 1$ and $q_2 = 0$. Then $q_2 = 0 \geq d_2$ which implies $d_2 = 0$. Hence $d_1 = 1 \leq q_1$ and one obtains the equation

$$\begin{aligned} M(q_1, q_2) &= \min\{M(q_1 - d_1, q_2 - d_2) + \mathit{colcost}(s_1[q_1], -) \mid (d_1, d_2) \in \{(1, 0)\}\} \\ &= M(q_1 - 1, q_2) + \mathit{colcost}(s_1[q_1], -) \end{aligned}$$

Now assume $q_1 = 0$ and $q_2 \geq 1$. In analogy to the previous case we get

$$M(q_1, q_2) = M(q_1, q_2 - 1) + \mathit{colcost}(-, s_2[q_2])$$

Finally, for $q_1 = 0$ and $q_2 = 0$ one obtains the equation $M(q_1, q_2) = 0$. Now put all four cases together, using the compact notation:

$$M(q_1, q_2) = \min \left\{ \begin{array}{ll} M(q_1 - 1, q_2) + \mathit{colcost}(s_1[q_1], -) & \text{if } q_1 \geq 1 \\ M(q_1, q_2 - 1) + \mathit{colcost}(-, s_2[q_2]) & \text{if } q_2 \geq 1 \\ M(q_1 - 1, q_2 - 1) + \mathit{colcost}(s_1[q_1], s_2[q_2]) & \text{if } q_1, q_2 \geq 1 \end{array} \right\}$$

Computing optimal multiple sequence alignments

- Let *colcost* be a function that assigns positive costs to pairs of different characters and cost 0 to pairs of identical characters.
- This holds for example for the consensus cost function as well as for the sum-of-pairs score function.
- Hence the previous equation, after renaming

M	q_1	q_2	s_1	s_2	*colcost*
E_δ	i	j	u	v	δ

leads to the equation

$$E_\delta(i,j) = \min \left\{ \begin{array}{ll} E_\delta(i-1,j) + \delta(u[i],-) & \text{if } i \geq 1 \\ E_\delta(i,j-1) + \delta(-,v[j]) & \text{if } j \geq 1 \\ E_\delta(i-1,j-1) + \delta(u[i],v[j]) & \text{if } i,j \geq 1 \end{array} \right\}$$

which is equivalent to the recurrence for E_δ shown at the beginning of this section. In an exercise you will be asked to partially evaluate the recurrence for $k = 3$.

Computing optimal multiple sequence alignments

Now let us analyze the efficiency of an algorithm computing M by

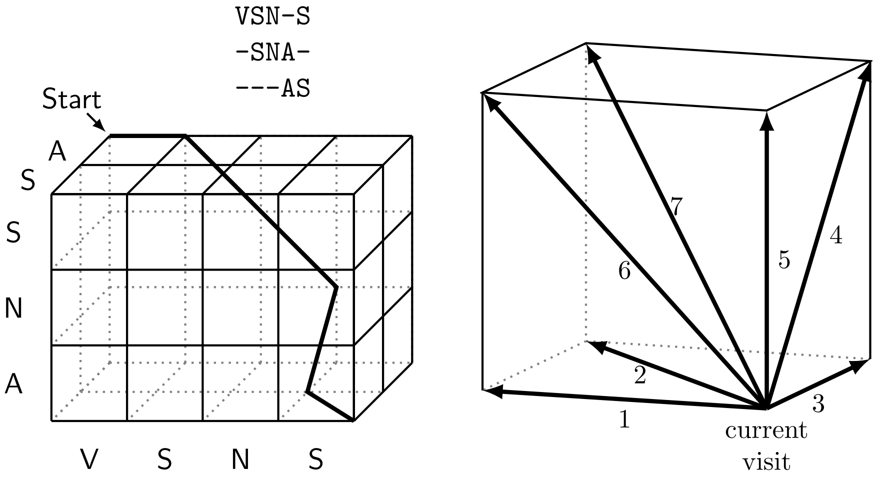
$$
\begin{aligned}
&M(q_1, \ldots, q_k) \\
&= \min\{M(q_1 - d_1, \ldots, q_k - d_k) + \\
&\qquad colcost(\varphi(q_1, d_1), \ldots, \varphi(q_k, d_k)) \mid (d_1, \ldots, d_k) \in \{0,1\} \times \cdots \times \{0,1\}, \\
&\qquad\qquad (d_1, \ldots, d_k) \neq (0, \ldots, 0), \\
&\qquad\qquad q_1 \geq d_1, \ldots, q_k \geq d_k\}
\end{aligned}
$$

- Table M has $z = \prod_{i=1}^{k}(n_i + 1)$ entries.
- Each of the z entries in table M requires to compute the minimum of up to $2^k - 1$ values, each corresponding to a k-tuple $(d_1, \ldots, d_k) \in \{0,1\} \times \cdots \times \{0,1\} \setminus \{(0, \ldots, 0)\}$.
- Each of the up to $2^k - 1$ values is the sum of another entry in table M plus the cost for the corresponding last column of an MSA.

Computing optimal multiple sequence alignments

- In case of the best consensus model, the column cost can be evaluated in $O(k)$ time.
- In case of the SP-scoring model, the column cost can be evaluated in $O(k^2)$ time, since all $k(k-1)/2$ pairwise scores have to be added up.
- In a geometric view, each MSA can be considered as a path in a k-dimensional hypercube from one corner to the opposite corner.
- See, for example, Figure 1 (left), showing the path corresponding to an MSA of the sequence `VSNS`, `SNA`, `AS`.
- The up to $2^k - 1$ dependencies of a single entry in table M correspond to $2^k - 1$ backward edges in the k-dimensional hypercube, see Figure 1 (right) showing $2^k - 1 = 2^3 = 7$ backward edges for $k = 3$.

Figure 1: The case for 3 sequences: alignment path (left) and 7 dependencies, each represented by an edge, for a given node (right) in the 3-dimensional table.



Computing optimal multiple sequence alignments

- Table M can be organized as a one dimensional array of size z.
- Each entry stores a distance value and a bitvector with k bits.
- The latter encodes a k-tuple $(d_1, \ldots, d_k)$ which gave rise to the minimum distance value.
- That is, bit i of the bitvector is set if and only if $d_i = 1$.
- Given $(q_1, \ldots, q_k)$, one computes the address of the entry in the array in $O(k)$ time, and accesses it in constant time.
- As a consequence, each entry is computed in $O(f(k) \cdot 2^k)$ time where $f(k) = k$ for the consensus model and $f(k) = k^2$ for the SP-scoring model.
- Hence the dynamic programming algorithm computes the distance of an optimal multiple alignment in $O(z \cdot f(k) \cdot 2^k)$ time.

Computing optimal multiple sequence alignments

- Having stored the bitvector in each entry (as described above), one can perform a traceback starting at entry $M(n_1, \dots, n_k)$, computing each column of an optimal alignment in $O(k)$ steps.
- Hence the traceback phase requires $O(k \cdot \ell)$ steps, where ℓ is the length of the *MSA*.
- The space requirement of the algorithm is $O(z)$.
- Given these results, it is clear that the method is only applicable, if k is small.

Example 6

Consider the SP-scoring model and assume that each computation step requires 1 nanosecond (i.e. 10^{-9} seconds). The following table shows the approximate running time (in years) to compute MSAs for k sequences, each of length $\ell = 100$ or $\ell = 200$:

	running times (years)	
k	$\ell = 100$	$\ell = 200$
7	$2.0 \cdot 10^0$	$2.6 \cdot 10^2$
8	$8.1 \cdot 10^2$	$2.1 \cdot 10^5$
9	$1.8 \cdot 10^5$	$9.1 \cdot 10^7$
10	$2.6 \cdot 10^7$	$2.7 \cdot 10^{10}$

Combining pairwise alignments

- Since the exact computation of optimal *MSA*s is very costly in practice, we will discuss methods which deliver approximations of the optimal solutions.
- The idea is to construct an *MSA* from a set of pairwise alignments of the given sequences.
- The resulting *MSA* of all sequences needs to incorporate the information from the pairwise alignments.
- This is expressed by the notion of compatibility, defined next.

Combining pairwise alignments

Let $S = \{s_1, \ldots, s_k\}$ be a set of sequences, and S' be a subset of S. Consider an *MSA* $Al_S = (s'_1, \ldots, s'_k)$ of S. The *projection of* Al_S *with respect to* S' is the *MSA* $proj(Al_S, S')$ produced as follows:

– delete all rows in Al_S which do not correspond to a sequence in S'.

– in the remaining rows delete all columns which only consist of gaps.

The alignment Al_S is said to be *compatible* with an alignment $Al_{S'}$ of S' if $Al_{S'} = proj(Al_S, S')$.

Example 7

Let $S = \{\mathtt{ACGG}, \mathtt{ATG}, \mathtt{ATCGG}\}$, $S' = \{\mathtt{ACGG}, \mathtt{ATG}\}$, $S'' = \{\mathtt{ATG}, \mathtt{ATCGG}\}$ and consider the following *MSA* Al_S of S:

```
A - C G G
A - - T G
A T C G G
```

Then $proj(Al_S, S')$ is as follows (so Al_S is compatible with this alignment):

```
A C G G
A - T G
```

And $proj(Al_S, S'')$ is as follows (so Al_S is compatible with this alignment):

```
A - - T G
A T C G G
```

Combining pairwise alignments

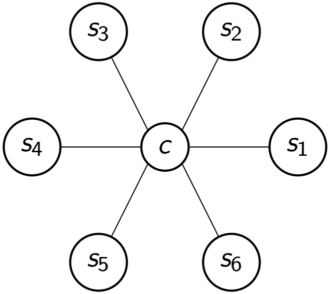
The relation between pairs of sequences can be represented by an alignment tree:

Definition 2

Let S be a set of sequences. An *alignment tree* for S is a labeled undirected tree where the node set is S and each edge (s, t) in the tree is labeled by an optimal pairwise alignment of the sequences s and t.

- There are methods which take an alignment tree, and construct a multiple alignment of S that is compatible with each of the optimal pairwise alignments in the tree.
- We will not discuss the general method to do this, but we will consider a special case where the alignment tree is a star, i.e. there is a center node c (the root) and edges (c, s) for all $s \in S \setminus \{c\}$ (which are the leaves).
- In this case, one often uses the notion *star alignment*.
- See also Figure 2 for a star alignment tree.

Figure 2: A star alignment tree of 7 sequences $c, s_1, \ldots, s_6$ with the center c and the remaining sequences $s_1, \ldots, s_6$ as leaves. Each edge from node s to node s' is implicitly labeled by an optimal pairwise alignment of s and s'.



Combining pairwise alignments

- The star alignment method first identifies the sequence in the center, and then constructs from the optimal pairwise alignments on the edges a multiple alignment that is compatible with the pairwise alignments.
- Whenever we have constructed a multiple alignment from the sequences $c, s_1, \ldots, s_i$, and we want to add the pairwise optimal alignment of c and s_{i+1}, we adhere to the principle "once a gap always a gap".
- That is, the gaps in the pairwise alignment of c and s_{i+1} are inherited to the multiple alignment, see Algorithm 1.
- In this way, the resulting multiple alignment will be compatible with the pairwise alignments.

Algorithm 1 (Star alignment)

Input: A set S of sequences
Output: Star alignment Al of S
Part (1): Compute the center sequence c of the star alignment

```
1: for all $s \in S$ do
2:     for all $t \in S \setminus \{s\}$ do
3:         compute an opt. alignment of $s$ and $t$ with dist. $\delta_{\text{opt}}(s,t)$
4:     end for
5: end for
6: $totalscore(t) \leftarrow \sum_{s \in S \setminus \{t\}} \delta_{\text{opt}}(s,t)$
7: let $c \in S$ s.t. $totalscore(c) \leq totalscore(s)$ for all $s \in S$
8: let $T$ be the star alignment tree with center $c$ and leaves $S \setminus \{c\}$
```

Algorithm 1 (Star alignment)

Part (2): Determine a compatible multiple alignment

```
1: choose an arbitrary $s \in S$
2: **let** $Al$ be an optimal alignment of $c$ and $s$
3: $S' \leftarrow \{c, s\}$
4: **while** $S' \neq S$ **do**
5:     choose an arbitrary $s' \in S \setminus S'$
6:     $S' \leftarrow S' \cup \{s'\}$
7:     Determine an $MSA$ $Al'$ of $S'$ by combining $Al$ with an optimal alignment
       of $c$ and $s'$, adhering to the principle "once a gap always a gap".
           ▷ assertion: $Al'$ is compatible with $Al$ and with the chosen optimal
       alignment of $c$ and $s'$.
8:     $Al \leftarrow Al'$
9: **end while**
```

Example 8

Consider the sequences $c = \texttt{ATGCATT}$, $s_1 = \texttt{AGTCAAT}$, $s_2 = \texttt{TCTCA}$, $s_3 = \texttt{ACTGTAATT}$ and the alignments

$$\begin{array}{rl} c' = & \texttt{A T G - C A T T} \\ s_1' = & \texttt{A - G T C A A T} \\ s_2' = & \texttt{- T C T C A - -} \end{array} \qquad \text{and} \qquad \begin{array}{rl} c'' = & \texttt{A - T G C - A T T} \\ s_3'' = & \texttt{A C T G T A A T T} \end{array}$$

The following table shows for which pairs of sequences the gaps are combined. The resulting *MSA* is shown in the last column.

combine		result
c'	c''	$c''' = \texttt{A-TG-C-ATT}$
s_1'	c''	$s_1''' = \texttt{A--GTC-AAT}$
s_2'	c''	$s_2''' = \texttt{--TCTC-A--}$
s_3''	c'	$s_3''' = \underbrace{\texttt{ACTG-TAATT}}_{\text{MSA}}$

Note that the resulting *MSA* is not optimal, since it would probably decrease the distance, if we would replace s_3''' by ACTGT-AATT.

Combining pairwise alignments

Next we show that Algorithm 1 well approximates the optimal SP-alignment, if the pairwise scoring function satisfies some additional properties.
We say that $\delta : \mathcal{A}' \times \mathcal{A}' \to \mathbb{R}_{\geq 0}$ is *good* if the following holds:

1. For each $a \in \mathcal{A}'$, we have $\delta(a, a) = 0$
2. For each $a, b, c \in \mathcal{A}'$, the triangle inequality $\delta(a, c) \leq \delta(a, b) + \delta(b, c)$ holds.

Combining pairwise alignments

Now we can prove the central theorem of this section:

Theorem 3

- *Let δ be a good cost function, let δ_{SP} be the function which evaluates the SP-score for a given MSA.*
- *Consider a set $S = \{s_1, \ldots, s_k\}$ of sequences, and the MSA $Al = (s'_1, \ldots, s'_k)$ of S, delivered by Algorithm 1.*
- *Then the following property holds:*

$$\delta_{\mathsf{SP}}\underbrace{(s'_1, \ldots, s'_k)}_{\substack{\textit{star}\\\textit{alignment}}} \leq \underbrace{\left(2 - \frac{2}{k}\right)}_{\substack{\textit{approx}\\\textit{factor}}} \cdot \underbrace{\delta_{\mathsf{opt-SP}}(s_1, \ldots, s_k)}_{\substack{\textit{optimal SP}\\\textit{alignment cost}}}$$

Combining pairwise alignments

- This result shows that the star alignment algorithm delivers a good approximation of the optimal SP alignment (by a factor $\leq \left(2 - \frac{2}{k}\right)$).
- By definition we also have $\delta_{\mathsf{opt-SP}}(s_1, \ldots, s_k) \leq \delta_{\mathsf{SP}}(s'_1, \ldots, s'_k)$ i.e. δ_{SP} is an upper bound for the optimal SP-alignment score.
- This can be exploited to reduce the search space for an optimal SP-alignment.
- To understand this, first note that we can efficiently compute $\delta_{\mathsf{SP}}(s'_1, \ldots, s'_k)$ using Algorithm 1.
- Now, if a value $d = M(q_1, \ldots, q_k)$ satisfies $d > \delta_{\mathsf{SP}}(s'_1, \ldots, s'_k)$ then $d > \delta_{\mathsf{opt-SP}}(s_1, \ldots, s_k)$ and hence an optimal *MSA* represented by a path in M cannot cross $M(q_1, \ldots, q_k)$.
- Hence, we do not have to compute any entry in matrix M larger than

$$\delta_{\mathsf{SP}}(s'_1, \ldots, s'_k)$$

This concept of reducing the search space is described by Carillo and Lipman [Carillo and Lipman, 1988] and it is implemented in the program MSA.

Iterative multiple alignment

A more general approach than the star-alignment uses a rooted tree with the sequences $s_1, \ldots, s_k$ at the leaves. It then iteratively combines alignments along the paths of the tree. More specifically, such an iterative multiple alignment approach consists of the following steps (see Fig. 3):

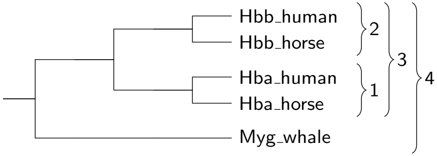
1. For all pairs of sequences, compute an optimal pairwise alignment and the corresponding edit distance.
2. From these pairwise distances compute a tree whose leaves are the sequences.
3. Successively combine pairwise alignments to larger and larger alignments, guided by the branching order of the tree until one large multiple alignment remains.

The mentioned tree is the so-called *guide tree*.

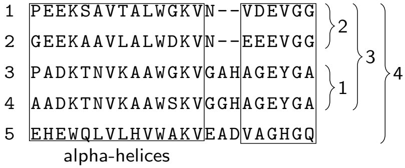
Figure 3: Overview of ClustalW. Figure adapted from [http://www.slideshare.net/jomcinerney/alignments]

	Hbb_human	Hbb_horse	Hba_human	Hba_horse	Myg_whale
Hbb_human	0.00				
Hbb_horse	0.17	0.00			
Hba_human	0.59	0.60	0.00		
Hba_horse	0.59	0.59	0.13	0.00	
Myg_whale	0.77	0.77	0.75	0.75	0.00

calculate matrix
of pairwise distances



compute guide tree
by upward clustering


```
1  PEEKSAVTALWGKVN--VDEVGG
2  GEEKAAVLALWDKVN--EEEVGG
3  PADKTNVKAAWGKVGAHAGEYGA
4  AADKTNVKAAWSKVGGHAGEYGA
5  EHEWQLVLHVWAKVEADVAGHGQ
```

iterative alignment
following guide tree

Iterative multiple alignment

- Suppose we have k sequences $s_1, \ldots, s_k$ and a distance $d(s_i, s_j)$ between each pair of sequences.
- A guide tree can be computed by clustering the given sequences, at each stage merging two sequences, and at the same time creating a new node in the tree.
- The tree is assembled “upwards”, first clustering pairs of leaves, then pairs of clustered leaves etc.
- Each node is given a height and the edge lengths are obtained as the difference of heights of its two end nodes.

This is exactly what the UPGMA algorithm does. It is explained in the next chapter on phylogenetic trees.

Iterative multiple alignment

The most striking advantages of the iterative multiple alignment method, in comparison to the optimal alignment computation via matrix M, are:

- This approach leads to fast algorithms that are applicable to many and long sequences.
- Conservations in subfamilies that are aligned before other, less closely related sequences are added, lead to high quality MSAs.

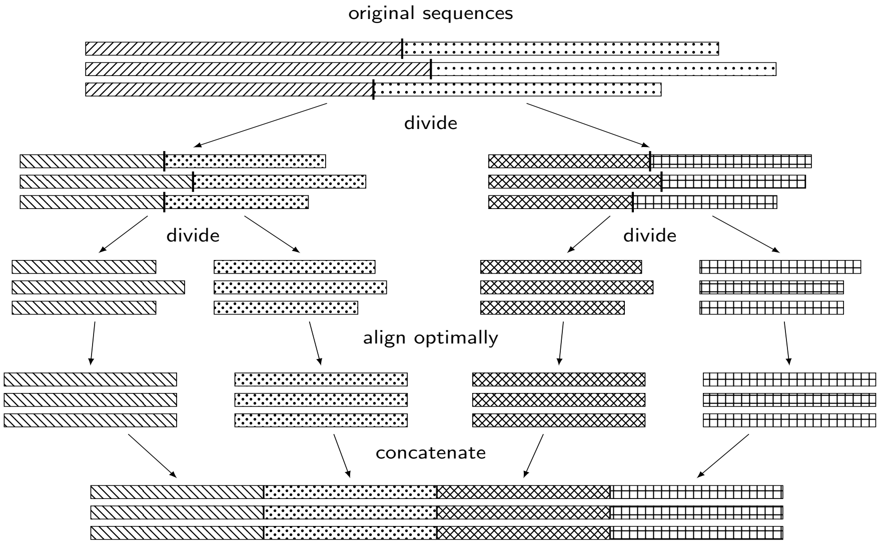
On the other hand there are a few disadvantages:

- Errors in an early step cannot be eliminated later, due to the applied principle "once a gap, always a gap".
- There is no well-defined objective function which must be optimized during the iterative alignment method. Thus the evaluation of the relative performance of this approach in comparison to other similar methods is difficult.
- A (possibly wrong) guide tree remains represented in the alignment and might bias the result.

Divide-and-conquer alignment

- The interdependence of tree and alignment is avoided in the divide-and-conquer multiple alignment algorithm (DCA, for short), described in [Stoye et al., 1997].
- It works for the sum-of-pairs cost model.
- The general idea of DCA is rather simple: Each sequence is cut in two after a suitable cut position somewhere close to its midpoint.
- This way, the problem of aligning one family of (long) sequences is divided into the two problems of aligning two families of (shorter) sequences, the prefix and the suffix sequences.
- This method is re-iterated until the sequences are sufficiently short, so that they can be aligned optimally by the program MSA.
- Finally, the resulting short alignments are concatenated, yielding a multiple alignment of the original sequences, see Figure 4 for a graphical explanation.

Figure 4: The divide-and-conquer alignment method (DCA)



Computing Cut Positions

- The main difficulty with this approach is to identify those cut-position combinations which lead to an optimal or (at least) close to optimal concatenated alignment.
- Here, a heuristic based on so-called additional-cost matrices is used.
- This heuristic quantifies the compatibility of cut positions in distinct sequences.

Definition 4

Given a sequence s of length n and a cut position c, $0 \leq c \leq n$, we denote by $s(\leq c)$ the prefix sequence $s[1 \ldots c]$ and by $s(> c)$ the suffix sequence $s[c+1 \ldots n]$.

Computing Cut Positions

Definition 5

For all pairs of sequences s_p, s_q and for all cut positions c_p of s_p and c_q of s_q, define

$$C_{s_p,s_q}[c_p, c_q] = \delta(s_p(\leq c_p), s_q(\leq c_q)) + \delta(s_p(> c_p), s_q(> c_q)) - \delta(s_p, s_q)$$

- $\delta(s_p, s_q)$ is an abbreviated notation for the edit distance of s_p and s_q with respect to the cost function δ.
- $\delta(s_p(\leq c_p), s_q(\leq c_q))$ is the edit distance of the two prefix sequences (ending at position c_p and c_q), respectively.
- $\delta(s_p(> c_p), s_q(> c_q))$ is the edit distance of the two suffix sequences (beginning at position $c_p + 1$ and $c_q + 1$), respectively.
- So $C_{s_p,s_q}[c_p, c_q]$ is the additional cost of forcing the pairwise alignment to not cross the cut positions c_p and c_q: If there is an optimal alignment which aligns $s_p(\leq c_p)$ with $s_q(\leq c_q)$ and $s_p(> c_p)$ with $s_q(> c_q)$, then $C_{s_p,s_q}[c_p, c_q] = 0$.

Computing Cut Positions

- One uses standard methods for computing the global edit distance of two sequences to determine the required values above.
- Next one combines these values to quantify the effect of choosing the cut positions for all k sequences.

Definition 6

For given cut positions $c_1, c_2, \ldots, c_k$ define

$$C(c_1, c_2, \ldots, c_k) = \sum_{p=1}^{k-1} \sum_{q=p+1}^{k} C_{s_p, s_q}[c_p, c_q]$$

- This is the additional cost for choosing the cut positions $c_1, c_2, \ldots, c_k$.
- There are heuristic algorithms which try to compute cut positions $c_1, c_2, \ldots, c_k$ minimizing $C(c_1, c_2, \ldots, c_k)$.

Computing Cut Positions

- Assume a function *ccp* which computes cut positions for a given set of sequences.
- Let Z be the cutoff value for the sequence lengths, that is, if the set of sequences to align contains a sequence of length $\leq Z$, then one applies the program MSA of Carillo and Lipman to the sequences to obtain an optimal multiple sequence alignment.
- The DCA algorithm can be described by the following recursive-function:

$$\begin{aligned}
DCA(s_1, s_2, \ldots, s_k) = {} & \text{if } |s_i| \leq Z \text{ for some } i, 1 \leq i \leq k \\
& \text{then } \mathbf{return}\ MSA(s_1, s_2, \ldots, s_k) \\
& \text{else } \mathbf{return}\ \ DCA(s_1(\leq c_1), s_2(\leq c_2), \ldots, s_k(\leq c_k)) \cdot \\
& \qquad\qquad\quad\ DCA(s_1(> c_1), s_2(> c_2), \ldots, s_k(> c_k)) \\
& \qquad\qquad\quad\ \mathbf{where}\ (c_1,c_2,\ldots,c_k) \leftarrow ccp(s_1,s_2,\ldots,s_k)
\end{aligned}$$

The operator $\cdot$ between the results of the recursive calls of DCA is the concatenation of two MSAs.

Carillo, H. and Lipman, D. (1988).
The Multiple Sequence Alignment Problem in Biology.
SIAM Journal of Applied Mathematics, 48(5):1073–1082.

Stoye, J., Moulton, V., and Dress, A. (1997).
DCA: An Efficient Implementation of the Divide-and-Conquer Approach to Simultaneous Multiple Sequence Alignment.
Comp. Appl. Biosci., 13(6).