Sequence Alignment

Motivation: assess similarity of sequences and learn about their evolutionary relationship

*Why do we want to know this?*

Example: *Sequences*  
ACCCGA  
ACTA  
TCCTA  

⇒align

*Alignment*  
ACCCGA  
AC--TA  
TCC--TA

Homology: Alignment reasonable, if sequences homologous

---

Definition (Sequence Homology)

Two or more sequences are *homologous* iff they evolved from a common ancestor.
Plan (and Some Preliminaries)

• First: study only pairwise alignment.
  Fix alphabet $\Sigma$, such that $- \notin \Sigma$. $-$ is called the gap symbol. The elements of $\Sigma^*$ are called sequences.
  Fix two sequences $a, b \in \Sigma^*$.

• For pairwise sequence comparison: define edit distance, define alignment distance, show equivalence of distances, define alignment problem and efficient algorithm
  gap penalties, local alignment

• Later: extend pairwise alignment to multiple alignment

Definition (Alphabet, words)

An alphabet $\Sigma$ is a finite set (of symbols/characters). $\Sigma^+$ denotes the set of non-empty words of $\Sigma$, i.e. $\Sigma^+ := \bigcup_{i>0} \Sigma^i$. A word $x \in \Sigma^n$ has length $n$, written $|x|$. $\Sigma^* := \Sigma^+ \cup \{\epsilon\}$, where $\epsilon$ denotes the empty word of length 0.
Levenshtein Distance

Definition
The *Levenshtein Distance* between two words/sequences is the minimal number of substitutions, insertions and deletions to transform one into the other.

Example
ACCCGA and ACTA have (at most) distance 3:
ACCCGA $\rightarrow$ ACCGA $\rightarrow$ ACCTA $\rightarrow$ ACTA

*In biology, operations have different cost. (Why?)*
Edit Distance: Operations

Definition (Edit Operations)
An *edit operation* is a pair \((x, y) \in (\Sigma \cup \{-\}) \neq (-, -)\). We call \((x, y)\)

- *substitution* iff \(x \neq -\) and \(y \neq -\)
- *deletion* iff \(y = -\)
- *insertion* iff \(x = -\)

For sequences \(a, b\), write \(a \rightarrow_{(x, y)} b\), iff \(a\) is transformed to \(b\) by
operation \((x, y)\). Furthermore, write \(a \Rightarrow_S b\), iff \(a\) is transformed
to \(b\) by a sequence of edit operations \(S\).

Example
\[
\begin{align*}
\text{ACCCGA} & \rightarrow_{(C, -)} \text{ACCGA} \rightarrow_{(G, T)} \text{ACCTA} \rightarrow_{(-, T)} \text{ATCCTA} \\
\text{ACCCGA} & \Rightarrow_{(C, -),(G, T),(-, T)} \text{ATCCTA}
\end{align*}
\]

Recall: \(- \notin \Sigma\), \(a, b\) are sequences in \(\Sigma^*\)
Definition (Cost, Edit Distance)

Let $w : (\Sigma \cup \{-\})^2 \rightarrow \mathbb{R}$, such that $w(x, y)$ is the cost of an edit operation $(x, y)$. The cost of a sequence of edit operations $S = e_1, \ldots, e_n$ is

$$\tilde{w}(S) = \sum_{i=1}^{n} w(e_i).$$

The edit distance of sequences $a$ and $b$ is

$$d_w(a, b) = \min\{\tilde{w}(S) \mid a \Rightarrow_S b\}.$$
Edit Distance: Cost and Problem Definition

Definition (Cost, Edit Distance)

Let \( w : (\Sigma \cup \{-\})^2 \to \mathbb{R} \), such that \( w(x, y) \) is the \textit{cost of an edit operation} \((x, y)\). The \textit{cost of a sequence of edit operations} \( S = e_1, \ldots, e_n \) is

\[
\tilde{w}(S) = \sum_{i=1}^{n} w(e_i).
\]

The \textit{edit distance of sequences} \( a \) and \( b \) is

\[
d_w(a, b) = \min\{\tilde{w}(S) \mid a \Rightarrow_S b\}.
\]

Is the definition reasonable?

Definition (Metric)

A function \( d : X^2 \to \mathbb{R} \) is called \textit{metric} iff 1.) \( d(x, y) = 0 \) iff \( x = y \) 2.) \( d(x, y) = d(y, x) \) 3.) \( d(x, y) \leq d(x, z) + d(z, y) \).

Remarks: 1.) for metric \( d \), \( d(x, y) \geq 0 \), 2.) \( d_w \) is metric iff \( w(x, y) \geq 0 \), 3.) In the following, assume \( d_w \) is metric.
Definition (Cost, Edit Distance)

Let \( w : (\Sigma \cup \{-\})^2 \to \mathbb{R} \), such that \( w(x, y) \) is the cost of an edit operation \((x, y)\). The cost of a sequence of edit operations \( S = e_1, \ldots, e_n \) is

\[
\tilde{w}(S) = \sum_{i=1}^{n} w(e_i).
\]

The edit distance of sequences \( a \) and \( b \) is

\[
d_w(a, b) = \min\{\tilde{w}(S) \mid a \Rightarrow_S b\}.
\]

Remarks

- Natural ’evolution-motivated’ problem definition.
- Not obvious how to compute edit distance efficiently
  \( \Rightarrow \) define alignment distance
Alignment Distance

Definition (Alignment)

A pair of words \(a^\diamond, b^\diamond \in (\Sigma \cup \{-\})^*\) is called alignment of sequences \(a\) and \(b\) (\(a^\diamond\) and \(b^\diamond\) are called alignment strings), iff

1. \(|a^\diamond| = |b^\diamond|\)
2. for all \(1 \leq i \leq |a^\diamond|: a_i^\diamond \neq -\) or \(b_i^\diamond \neq -\)
3. deleting all gap symbols \(-\) from \(a^\diamond\) yields \(a\) and deleting all \(-\) from \(b^\diamond\) yields \(b\)

Example

\(a = \text{ACGGAT}\)
\(b = \text{CCGCTT}\)

possible alignments are

\(a^\diamond = \text{AC-GG-AT}\) or \(a^\diamond = \text{ACGG---AT}\) or \(b^\diamond = \text{--CCGCT-T}\) or \(b^\diamond = \text{--CCGCT-T}\) or \ldots\) (exponentially many)

edit operations of first alignment: \((A,-),(\cdot,C),(G,C),(\cdot,T),(A,-)\)
Definition (Cost of Alignment, Alignment Distance)

The cost of the alignment \((a^\diamond, b^\diamond)\), given a cost function \(w\) on edit operations is

\[
w(a^\diamond, b^\diamond) = \sum_{i=1}^{\|a^\diamond\|} w(a_i^\diamond, b_i^\diamond)
\]

The alignment distance of \(a\) and \(b\) is

\[
D_w(a, b) = \min\{w(a^\diamond, b^\diamond) \mid (a^\diamond, b^\diamond) \text{ is alignment of } a \text{ and } b\}.
\]
Theorem (Equivalence of Edit and Alignment Distance)

For metric $w$, $d_w(a, b) = D_w(a, b)$.

Recall:

Definition (Edit Distance)

The *edit distance of $a$ and $b$* is

$$d_w(a, b) = \min\{\tilde{w}(S) \mid a\text{ transformed to } b\text{ by e.o.-sequence } S\}.$$

Definition (Alignment Distance)

The *alignment distance* of $a$ and $b$ is

$$D_w(a, b) = \min\{w(\cdot, \cdot) \mid (\cdot, \cdot)\text{ is alignment of } a\text{ and } b\}.$$
Alignment Distance $= \text{Edit Distance}$

**Theorem (Equivalence of Edit and Alignment Distance)**

For metric $w$, $d_w(a, b) = D_w(a, b)$.

**Remarks**

- Proof idea:
  
  $d_w(a, b) \leq D_w(a, b)$: alignment yields sequence of edit ops
  
  $D_w(a, b) \leq d_w(a, b)$: sequence of edit ops yields equal or better alignment (needs triangle inequality)

- Reduces edit distance to alignment distance

- We will see: the alignment distance is computed efficiently by dynamic programming (using *Bellman’s Principle of Optimality*).
Principle of Optimality and Dynamic Programming

Principle of Optimality:
‘Optimal solutions consist of optimal partial solutions’

Example: Shortest Path

Idea of Dynamic Programming (DP):
• Solve partial problems first and materialize results
• (recursively) solve larger problems based on smaller ones

Remarks
• The principle is valid for the alignment distance problem
• Principle of Optimality enables the programming method DP
• Dynamic programming is widely used in Computational Biology and you will meet it quite often in this class
Alignment Matrix

Idea: choose alignment distances of prefixes \(a_{1..i}\) and \(b_{1..j}\) as partial solutions and define matrix of these partial solutions.

Let \(n := |a|, m := |b|\).

Definition (Alignment matrix)
The alignment matrix of \(a\) and \(b\) is the \((n + 1) \times (m + 1)\)-matrix \(D := (D_{ij})_{0 \leq i \leq n, 0 \leq j \leq m}\) defined by

\[D_{ij} := D_w(a_{1..i}, b_{1..j})\]

\(= \min\{w(a^\diamond, b^\diamond) \mid (a^\diamond, b^\diamond) \text{ is alignment of } a_{1..i} \text{ and } b_{1..j}\}\).

Notational remarks

- \(a_i\) is the \(i\)-th character of \(a\)
- \(a_{x..y}\) is the sequence \(a_x a_{x+1} \ldots a_y\) (subsequence of \(a\)).
- by convention \(a_{x..y} = \epsilon\) if \(x > y\).
Alignment Matrix Example

Example

- \( a = \text{AT}, \ b = \text{AAGT} \)
- \( w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
1 & \text{otherwise}
\end{cases} \)

Remark: The alignment matrix \( D \) contains the alignment distance (=edit distance) of \( a \) and \( b \) in \( D_{n,m} \).
Alignment Matrix Example

Example

- \( a = \text{AT}, \ b = \text{AAGT} \)
- \( w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
1 & \text{otherwise} 
\end{cases} \)

\[
\begin{array}{cccccc}
& A & A & G & T \\
A & 0 & 1 & 2 & 3 & 4 \\
T & 1 & 0 & 1 & 2 & 3 \\
\end{array}
\]

Remark: The alignment matrix \( D \) contains the alignment distance (\(=\)edit distance) of \( a \) and \( b \) in \( D_{n,m} \).
Needleman-Wunsch Algorithm

Claim
For \((a^\diamond, b^\diamond)\) alignment of \(a\) and \(b\) with length \(r = |a^\diamond|\),

\[ w(a^\diamond, b^\diamond) = w(a_{1..r-1}^\diamond, b_{1..r-1}^\diamond) + w(a_r^\diamond, b_r^\diamond). \]

Theorem
For the alignment matrix \(D\) of \(a\) and \(b\), holds that

- \(D_{0,0} = 0\)
- for all \(1 \leq i \leq n\): \(D_{i,0} = \sum_{k=1}^{i} w(a_k, -) = D_{i-1,0} + w(a_i, -)\)
- for all \(1 \leq j \leq m\): \(D_{0,j} = \sum_{k=1}^{j} w(-, b_k) = D_{0,j-1} + w(-, b_j)\)
- \(D_{ij} = \min\left\{ \begin{array}{ll} D_{i-1,j-1} + w(a_i, b_j) & \text{(match)} \\ D_{i-1,j} + w(a_i, -) & \text{(deletion)} \\ D_{i,j-1} + w(-, b_j) & \text{(insertion)} \end{array} \right. \)

Remark: The theorem claims that each prefix alignment distance can be computed from a constant number of smaller ones.

Proof ????
Needleman-Wunsch Algorithm

Claim
For \((a^\diamond, b^\diamond)\) alignment of \(a\) and \(b\) with length \(r = |a^\diamond|\),
\[
w(a^\diamond, b^\diamond) = w(a_{1..r-1}^\diamond, b_{1..r-1}^\diamond) + w(a_r^\diamond, b_r^\diamond).
\]

Theorem
For the alignment matrix \(D\) of \(a\) and \(b\), holds that
\[D_{0,0} = 0\]
\[\text{for all } 1 \leq i \leq n: D_{i,0} = \sum_{k=1}^{i} w(a_k, -) = D_{i-1,0} + w(a_i, -)\]
\[\text{for all } 1 \leq j \leq m: D_{0,j} = \sum_{k=1}^{j} w(-, b_k) = D_{0,j-1} + w(-, b_j)\]
\[D_{ij} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & \text{ (match)} \\ D_{i-1,j} + w(a_i, -) & \text{ (deletion)} \\ D_{i,j-1} + w(-, b_j) & \text{ (insertion)} \end{cases} \]

Remark: The theorem claims that each prefix alignment distance can be computed from a constant number of smaller ones.

Proof: Induction over \(i+j\)
Needleman-Wunsch Algorithm (Pseudocode)

\[ D_{0,0} := 0 \]

\textbf{for} \quad i := 1 \quad \textbf{to} \quad n \quad \textbf{do}

\[ D_{i,0} := D_{i-1,0} + w(a_i, -) \]

\textbf{end for}

\textbf{for} \quad j := 1 \quad \textbf{to} \quad m \quad \textbf{do}

\[ D_{0,j} := D_{0,j-1} + w(-, b_j) \]

\textbf{end for}

\textbf{for} \quad i := 1 \quad \textbf{to} \quad n \quad \textbf{do}

\textbf{for} \quad j := 1 \quad \textbf{to} \quad m \quad \textbf{do}

\[ D_{i,j} := \min \left\{ D_{i-1,j-1} + w(a_i, b_j), D_{i-1,j} + w(a_i, -), D_{i,j-1} + w(-, b_j) \right\} \]

\textbf{end for}

\textbf{end for}
Example

- \( a = AT, \ b = AAGT \)
- \( w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
1 & \text{otherwise} 
\end{cases} \)

Open: how to find best alignment?
Example

- $a = AT$, $b = AAGT$
- $w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
1 & \text{otherwise}
\end{cases}$

Open: how to find best alignment?
Traceback

\[ w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
1 & \text{otherwise} 
\end{cases} \]

A A G T

0 1 2 3 4

A

1 0 1 2 3

T

2 1 1 2 2

Remarks

- Start in \((n, m)\). For every \((i, j)\) determine optimal case.
- Not necessarily unique.
- Sequence of \textit{trace arrows} let's infer best alignment.
Introduction

Traceback

\[ w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
1 & \text{otherwise} 
\end{cases} \]

Remarks

- Start in \((n, m)\). For every \((i, j)\) determine optimal case.
- Not necessarily unique.
- Sequence of \textit{trace arrows} let’s infer best alignment.
Complexity

- compute one entry: three cases, i.e. constant time
- \( nm \) entries \( \Rightarrow \) fill matrix in \( O(nm) \) time
- traceback: \( O(n + m) \) time
- TOTAL: \( O(n^2) \) time and space (assuming \( m \leq n \))

Remarks

- assuming \( m \leq n \) is w.l.o.g. since we can exchange \( a \) and \( b \)
- space complexity can be improved to \( O(n) \) for computation of distance (simple, “store only current and last row”) and traceback (more involved; Hirschberg-algorithm uses “Divide and Conquer” for computing trace)
• We have seen how to compute the pairwise edit distance and the corresponding optimal alignment.
• Before going multiple, we will look at two further special topics for pairwise alignment:
  • more realistic, non-linear gap cost and
  • similarity scores and local alignment
Alignment Cost Revisited

Motivation:

- The alignments \( GA--T \) and \( G-A-T \) have the same edit distance.
- The first one is biologically more reasonable: it is more likely that evolution introduces one large gap than two small ones.
- This means: gap cost should be non-linear, sub-additive!
**Gap Penalty**

**Definition (Gap Penalty)**

A *gap penalty* is a function $g : \mathbb{N} \rightarrow \mathbb{R}$ that is sub-additive, i.e.

$$g(k + l) \leq g(k) + g(l).$$

A *gap* in an alignment string $a^\diamond$ is a substring of $a^\diamond$ that consists of only gap symbols — and is maximally extended. $\Delta a^\diamond$ is the multi-set of gaps in $a^\diamond$.

The *alignment cost with gap penalty $g$ of $(a^\diamond, b^\diamond)$* is

$$w_g(a^\diamond, b^\diamond) = \sum_{1 \leq r \leq |a^\diamond|, \text{ where } a^\diamond_r \neq -, b^\diamond_r \neq -} w(a^\diamond_r, b^\diamond_r) \quad (\text{cost of mismatches})$$

$$+ \sum_{x \in \Delta a^\diamond \cup \Delta b^\diamond} g(|x|) \quad (\text{cost of gaps})$$

**Example:**

$$a^\diamond = \text{ATG---CGAC--GC} \quad \Rightarrow \quad \Delta a^\diamond = \{---, --\}, \quad \Delta b^\diamond = \{-, -\}$$

$$b^\diamond = \text{-TGCGGCG-CTTTC}$$
General sub-additive gap penalty

Theorem

Let \( D \) be the alignment matrix of \( a \) and \( b \) with cost \( w \) and gap penalty \( g \), such that \( D_{ij} = w_g(a_{1..i}, b_{1..j}) \). Then:

- \( D_{0,0} = 0 \)
- \( \text{for all } 1 \leq i \leq n: \ D_{i,0} = g(i) \)
- \( \text{for all } 1 \leq j \leq m: \ D_{0,j} = g(j) \)

\[
D_{ij} = \min \begin{cases} 
  D_{i-1,j-1} + w(a_i, b_j) & \text{(match)} \\
  \min_{1 \leq k \leq i} D_{i-k,j} + g(k) & \text{(deletion of length } k) \\
  \min_{1 \leq k \leq j} D_{i,j-k} + g(k) & \text{(insertion of length } k) 
\end{cases}
\]

Remarks

- Complexity \( O(n^3) \) time, \( O(n^2) \) space
- pseudocode, correctness, traceback left as exercise
- much more realistic, but significantly more expensive than Needleman-Wunsch \( \Rightarrow \) can we improve it?
Affine gap cost

Definition
A gap penalty is affine, iff there are real constants $\alpha$ and $\beta$, such that for all $k \in \mathbb{N}$: $g(k) = \alpha + \beta k$.

Remarks

- Affine gap penalties almost as good as general ones: Distinguishing gap opening ($\alpha$) and gap extension cost ($\beta$) is “biologically reasonable”.
- The minimal alignment cost with affine gap penalty can be computed in $O(n^2)$ time! (Gotoh algorithm)
Gotoh algorithm: sketch only

In addition to the alignment matrix $D$, define two further matrices/states.

- $A_{i,j} :=$ cost of best alignment of $a_{1..i}, b_{1..j}$, that ends with deletion $a_i$.
- $B_{i,j} :=$ cost of best alignment of $a_{1..i}, b_{1..j}$, that ends with insertion $b_j$.

**Recursions:**

\[
A_{i,j} = \min \begin{cases} 
A_{i-1,j} + \beta \\
D_{i-1,j} + g(1)
\end{cases} \quad \text{(deletion extension)}
\]

\[
B_{i,j} = \min \begin{cases} 
B_{i,j-1} + \beta \\
D_{i,j-1} + g(1)
\end{cases} \quad \text{(insertion extension)}
\]

\[
D_{ij} = \min \begin{cases} 
D_{i-1,j-1} + w(a_i, b_j) \\
A_{i,j} \quad \text{(match)} \\
B_{i,j} \quad \text{(deletion closing)} \\
\end{cases}
\]

\[
\text{Remark: } O(n^2) \text{ time and space}
\]
Similarity

Definition (Similarity)

The similarity of an alignment \((a^\diamond, b^\diamond)\) is

\[
s(a^\diamond, b^\diamond) = \sum_{i=1}^{\lvert a^\diamond \rvert} s(a_i^\diamond, b_i^\diamond),
\]

where \(s : (\Sigma \cup \{\cdot\})^2 \rightarrow \mathbb{R}\) is a similarity function, where for \(x \in \Sigma : s(x, x) > 0, s(x, -) < 0, s(-, x) < 0\).

*Observation*. Instead of minimizing alignment cost, one can maximize similarity:

\[
S_{ij} = \max \begin{cases} 
S_{i-1,j-1} + s(a_i, b_j) \\
S_{i-1,j} + s(a_i, -) \\
S_{i,j-1} + s(-, b_j)
\end{cases}
\]

*Motivation:*

- defining similarity of 'building blocks' could be more natural, e.g. similarity of amino acids.
- similarity is useful for *local alignment*
Local Alignment Motivation

Local alignment asks for the best alignment of any two subsequences of $a$ and $b$. Important Application: Search! (e.g. BLAST combines heuristics and local alignment)

Example

$$a = \text{AWGVIACAILAGRS}$$
$$b = \text{VIVTAIAVAGYY}$$

In contrast, all previous methods compute “global alignments”. Why is distance not useful?

Example

a) $\text{XXXAAXXXX}$
   $\text{YYAAYY}$

b) $\text{XXAAAAAXXXX}$
   $\text{YYYAAAAAYY}$

Where is the stronger local motif? Only similarity can distinguish.
Local Alignment

Definition (Local Alignment Problem)
Let $s$ be a similarity on alignments.

$$S_{\text{global}}(a, b) := \max_{(\alpha^\circ, \beta^\circ)} s(\alpha^\circ, \beta^\circ) \quad (\text{global similarity})$$

alignment of $a$ and $b$

$$S_{\text{local}}(a, b) := \max_{1 \leq i' < i \leq n, 1 \leq j' < j \leq m} S_{\text{global}}(a_{i'..i}, b_{j'..j}) \quad (\text{local similarity})$$

The local alignment problem is to compute $S_{\text{local}}(a, b)$.

Remarks

- That is, local alignment asks for the subsequences of $a$ and $b$ that have the best alignment.
- How would we define the local alignment matrix for DP?
- For example, why does “$H_{i,j} := S_{\text{local}}(a_{1..i}, b_{1..j})$” not work?
Local Alignment Matrix

Definition
The *local alignment matrix* \( H \) of \( a \) and \( b \) is \( (H_{i,j})_{0 \leq i \leq n, 0 \leq j \leq m} \) defined by

\[
H_{i,j} := \max_{0 \leq i' \leq i, 0 \leq j' \leq j} S_{\text{global}}(a_{i'+1..i}, b_{j'+1..j}).
\]

Remarks

- \( S_{\text{local}}(a, b) = \max_{i,j} H_{i,j} \) (!)
- all entries \( H_{i,j} \geq 0 \), since \( S_{\text{global}}(\epsilon, \epsilon) = 0 \).
- \( H_{i,j} = 0 \) implies no subsequences of \( a \) and \( b \) that end in respective \( i \) and \( j \) are similar.
- Allows case distinction / Principle of optimality holds!
Local Alignment Algorithm — Case Distinction

Cases for $H_{i,j}$

1.) $\cdots \ a_i \ b_i \ \cdots$

2.) $\cdots \ a_i \ - \ \cdots$

3.) $\cdots \ - \ \cdots \ b_j$

4.) 0, since if each of the above cases is dissimilar (i.e. negative), there is still $(\epsilon, \epsilon)$. 
Local Alignment Algorithm (Smith-Waterman Algorithm)

Theorem
For the local alignment matrix $H$ of $a$ and $b$,

- $H_{0,0} = 0$
- for all $1 \leq i \leq n$: $H_{i,0} = 0$
- for all $1 \leq j \leq m$: $H_{0,j} = 0$
- $H_{ij} = \max \begin{cases} 0 & \text{(empty alignment)} \\ H_{i-1,j-1} + s(a_i, b_j) \\ H_{i-1,j} + s(a_i, -) \\ H_{i,j-1} + s(-, b_j) \end{cases}$
Local Alignment Remarks

Remarks

• Complexity $O(n^2)$ time and space, again space complexity can be improved
• Requires that similarity function is centered around zero, i.e. positive = similar, negative = dissimilar.
• Extension to affine gap cost works
• Traceback?
Local Alignment Example

Example

- $a = \text{AAC}$, $b = \text{ACAA}$
- $s(x, y) = \begin{cases} 
2 & \text{iff } x = y \\
-3 & \text{otherwise} 
\end{cases}$

Traceback: start at maximum entry, trace back to first 0 entry
Substitution/Similarity Matrices

- In practice: use similarity matrices learned from closely related sequences or multiple alignments
- PAM (Percent Accepted Mutations) for proteins
- BLOSUM (BLOcks of Amino Acid SUbstitution) for proteins
- RIBOSUM for RNA
- Scores are (scaled) log odd scores: $\log \frac{Pr[x,y|\text{Related}]}{Pr[x,y|\text{Background}]}$

For example, BLOSUM62:

<table>
<thead>
<tr>
<th></th>
<th>Ala</th>
<th>Arg</th>
<th>Asn</th>
<th>Asp</th>
<th>Cys</th>
<th>Gln</th>
<th>Glu</th>
<th>Gly</th>
<th>His</th>
<th>Ile</th>
<th>Leu</th>
<th>Lys</th>
<th>Met</th>
<th>Phe</th>
<th>Pro</th>
<th>Ser</th>
<th>Thr</th>
<th>Trp</th>
<th>Tyr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ala</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg</td>
<td>-1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asn</td>
<td>-2</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asp</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cys</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gln</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glu</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>-4</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gly</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>His</td>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ile</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-4</td>
<td>-3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leu</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-3</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lys</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>1</td>
<td>1</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Met</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phe</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pro</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ser</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thr</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trp</td>
<td>-3</td>
<td>-3</td>
<td>-4</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tyr</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>3</td>
<td>-3</td>
<td>-2</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Val</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-3</td>
<td>-1</td>
<td>4</td>
</tr>
</tbody>
</table>

Ala Arg Asn Asp Cys Gln Glu Gly His Ile Leu Lys Met Phe Pro Ser Thr Trp Tyr Val
Multiple Alignment

Example: Sequences

\[
\begin{align*}
a^{(1)} &= \text{ACCCGAG} \\
a^{(2)} &= \text{ACTACC} \\
a^{(3)} &= \text{TCCTACCGG}
\end{align*}
\]

⇒align

\[
\begin{align*}
\text{Alignment} &= \text{ACCCGA–G–} \\
&\quad \quad \text{AC--TAC–C} \\
&\quad \quad \text{TCC–TACGG}
\end{align*}
\]

Definition

A multiple alignment \( A \) of \( K \) sequences \( a^{(1)}...a^{(K)} \) is a \( K \times N \)-matrix \( (A_{i,j})_{1\leq i \leq K, 1\leq j \leq N} \) (\( N \) is the number of columns of \( A \)) where

1. each entry \( A_{i,j} \in (\Sigma \cup \{-\}) \)
2. for each row \( i \): deleting all gaps from \( (A_{i,1}...A_{i,N}) \) yields \( a^{(i)} \)
3. no column \( j \) contains only gap symbols
How to Score Multiple Alignments

As for pairwise alignment:

- Assume columns are scored independently
- Score is sum over alignment columns

\[
S(A) = \sum_{j=1}^{N} s(A_{1j}, \ldots, A_{Kj})
\]

Example

\[
S(A) = s\begin{pmatrix} A \\ A \\ T \end{pmatrix} + s\begin{pmatrix} C \\ C \\ C \end{pmatrix} + s\begin{pmatrix} C \end{pmatrix} + s\begin{pmatrix} C \end{pmatrix} + \cdots + s\begin{pmatrix} C \\ C \end{pmatrix}
\]

How do we know similarities?
How to Score Multiple Alignments

As for pairwise alignment:

- Assume columns are scored independently
- Score is sum over alignment columns

\[ S(A) = \sum_{j=1}^{N} s\left(\begin{array}{c} A_{1j} \\ \cdots \\ A_{Kj} \end{array}\right) \]

Example

\[ S(A) = s\left(\begin{array}{c} A \\ T \end{array}\right) + s\left(\begin{array}{c} C \\ C \end{array}\right) + s\left(\begin{array}{c} C \\ - \end{array}\right) + s\left(\begin{array}{c} - \\ - \end{array}\right) + \cdots + s\left(\begin{array}{c} - \\ C \end{array}\right) \]

How to define \( s\left(\begin{array}{c} x \\ y \\ z \end{array}\right) \)? as log odds \( s\left(\begin{array}{c} x \\ y \\ z \end{array}\right) = \log \frac{Pr[x,y,z|\text{Related}]}{Pr[x,y,z|\text{Background}}} \)?

Problems? Can we learn similarities for triples, 4-tuples, . . . ?
Sum-Of-Pairs Score

Idea: approximate column scores by pairwise scores

\[ s\left(\begin{array}{c} x_1 \\ \vdots \\ x_j \end{array}\right) = \sum_{1 \leq k < l \leq K} s(x_k, x_l) \]

Sum-of-pairs is the most commonly used scoring scheme for multiple alignments.
(Extensible to gap penalties, in particular affine gap cost)

Drawbacks?
**Optimal Multiple Alignment**

*Idea:* use dynamic programming

**Example**

For 3 sequences $a, b, c$, use 3-dimensional matrix (after initialization:)

$$S_{i,j,k} = \max \begin{cases} 
S_{i-1,j-1,k-1} + s(a_i, b_j, c_k) \\
S_{i-1,j-1,k} + s(a_i, b_j, -) \\
S_{i-1,j,k-1} + s(a_i, - , c_k) \\
S_{i,j-1,k-1} + s(-, b_j, c_k) \\
S_{i-1,j,k} + s(a_i, - , -) \\
S_{i,j-1,k} + s(-, b_j, -) \\
S_{i,j,k-1} + s(-, - , c_k)
\end{cases}$$

For K sequences use K-dimensional matrix.

*Complexity?*
Heuristic Multiple Alignment: Progressive Alignment

Idea: compute optimal alignments only pairwise

Example

4 sequences $a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}$

1. determine how they are related
   ⇒ tree, e.g. $(((a^{(1)}, a^{(2)}), (a^{(3)}, a^{(4)})))$

2. align most closely related sequences first
   ⇒ (optimally) align $a^{(1)}$ and $a^{(2)}$ by DP

3. go on ⇒ (optimally) align $a^{(3)}$ and $a^{(4)}$ by DP

4. go on?! ⇒ (optimally) align the two alignments
   How can we do that?

5. Done. We produced a multiple alignment of $a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}$.

Remarks: - Optimality is not guaranteed. Why?
- The tree is known as guide tree. How can we get it?
Guide tree

The guide tree determines the order of pairwise alignments in the progressive alignment scheme. The order of the progressive alignment steps is crucial for quality!

Heuristics:

1. Compute pairwise distances between all input sequences
   - align all against all
   - in case, transform similarities to distances (e.g. Feng-Doolittle)

2. Cluster sequences by their distances, e.g. by
   - Unweighted Pair Group Method (UPGMA)
   - Neighbor Joining (NJ)
Aligning Alignments

Two (multiple) alignments $A$ and $B$ can be aligned by DP in the same way as two sequences.

Idea:

- An alignment is a sequence of alignment columns.

\[
\begin{align*}
ACCCGAG- & \\
AC--TAC-C & \\
TCC-TACGG & \\
\end{align*}
\]

\[
\equiv \left( A \right) \left( C \right) \left( C \right) \left( - \right) \left( - \right) \ldots \left( C \right).
\]

- Assign similarity to two columns from resp. $A$ and $B$, e.g.

\[
s(\left( - \right), \left( G \right)) \text{ by } \text{sum-of-pairs}.
\]

We can use dynamic programming, which recurses over alignment scores of prefixes of alignments.

Consequences for progressive alignment scheme:

- Optimization only \textit{local}.
- Commits to local decisions. “Once a gap, always a gap”
Progressive Alignment — Example

**IN:** $a^{(1)} = A C C G$, $a^{(2)} = T T G G$, $a^{(3)} = T C G$, $a^{(4)} = C T G G$

$$w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
2 & \text{iff } x = - \text{ or } y = - \\
3 & \text{otherwise (for mismatch)} 
\end{cases}$$

- Compute all against all edit distances and cluster

<table>
<thead>
<tr>
<th>Align ACCG and TTGG</th>
<th>Align ACCG and TCG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Align ACCG and CTGG</th>
<th>Align TTGG and TCG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Align TTGG and CTGG</th>
<th>Align TCG and CTGG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>8</td>
</tr>
</tbody>
</table>
Progressive Alignment — Example

IN: \( a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG \)

\[ w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
2 & \text{iff } x = - \text{ or } y = - \\
3 & \text{otherwise (for mismatch)}
\end{cases} \]

- Compute all against all edit distances and cluster
  \[ \Rightarrow \text{distance matrix} \]

\[
\begin{array}{cccc}
  a^{(1)} & a^{(2)} & a^{(3)} & a^{(4)} \\
  a^{(1)} & 0 & 9 & 5 & 8 \\
  a^{(2)} & 0 & 5 & 3 \\
  a^{(3)} & 0 & 5 \\
  a^{(4)} & 0 \\
\end{array}
\]

\[ \Rightarrow \text{Cluster (e.g. UPGMA)} \]
\[ a^{(2)} \text{ and } a^{(4)} \text{ are closest, Then: } a^{(1)} \text{ and } a^{(3)} \]
Progressive Alignment — Example

IN: \(a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG\)

\[w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
2 & \text{iff } x = - \text{ or } y = - \\
3 & \text{otherwise (for mismatch)}
\end{cases}\]

- Compute all against all edit distances and cluster
  \(\Rightarrow\) guide tree \(((a^{(2)}, a^{(4)}), (a^{(1)}, a^{(3)}))\)

- Align \(a^{(2)}\) and \(a^{(4)}\): \(\begin{array}{c}
TTGG \\
CTGG
\end{array}\)
- Align \(a^{(1)}\) and \(a^{(3)}\): \(\begin{array}{c}
ACCG \\
-TCG
\end{array}\)

- Align the alignments!

\[
\begin{array}{cccccc}
& A & C & C & G \\
- & T & C & G \\
\hline
T & 0 & 4 & 12 & 20 & 28 \\
C & 8 & 10 & \cdots & \\
T & 16 & \cdots & \\
G & 24 & \\
G & 32 & \\
\end{array}
\]
Progressive Alignment — Example

IN: \( a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG \)

\[
 w(x, y) = \begin{cases} 
 0 & \text{iff } x = y \\
 2 & \text{iff } x = \_ \text{ or } y = \_ \\
 3 & \text{otherwise (for mismatch)} 
\end{cases}
\]

- Compute all against all edit distances and cluster
  \[ \Rightarrow \text{guide tree } ((a^{(2)}, a^{(4)}), (a^{(1)}, a^{(3)})) \]

- Align \( a^{(2)} \) and \( a^{(4)} \): \( TTGG \)
- Align \( a^{(1)} \) and \( a^{(3)} \): \( ACCG \)

- Align the alignments!

\[
\begin{array}{cccccc}
\text{Align} & \text{TTGG} & \text{and} & \text{ACCG} & \text{CTGG} \\
\text{TTGG} & A & C & C & G \\
\text{CTGG} & - & T & C & G \\
\text{TC} & 0 & 4 & 12 & 20 & 28 \\
\text{TT} & 8 & 10 & \ldots \\
\text{GG} & 16 & \ldots \\
\text{GG} & 24 & \ldots \\
\text{GG} & 32 & \ldots \\
\end{array}
\]

- \( w(TC, \_\_\_) = \)
  \[ w(T, \_) + w(C, \_) + w(T, \_) + w(C, \_) = 8 \]

- \( w(\_\_, A\_) = \)
  \[ w(\_, A) + w(\_, \_, \_) + w(\_, A) + w(\_, \_, \_) = 4 \]

- \( w(TC, A\_) = \)
  \[ w(T, A) + w(C, A) + w(T, \_) + w(C, \_) = 10 \]

- \( w(TC, CT) = \)
  \[ w(T, C) + w(C, C) + w(T, T) + w(C, T) = 6 \]

- \( \ldots \)
Progressive Alignment — Example

IN: $a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG$

$$w(x, y) = \begin{cases} 
0 & \text{iff } x = y \\
2 & \text{iff } x = - \text{ or } y = - \\
3 & \text{otherwise (for mismatch)}
\end{cases}$$

- Compute all against all edit distances and cluster
  $\Rightarrow$ guide tree $((a^{(2)}, a^{(4)}), (a^{(1)}, a^{(3)}))$
- Align $a^{(2)}$ and $a^{(4)}$: $\begin{array}{c|c|c|c} 
TTGG \\
CTGG 
\end{array}$
  Align $a^{(1)}$ and $a^{(3)}$: $\begin{array}{c|c|c|c} 
ACCG \\
-TCG 
\end{array}$
- Align the alignments!

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>C</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>TT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

$\Rightarrow$ after filling and traceback

```
TTGG
CTGG
ACCG
-TCG
```
A Classical Approach: CLUSTAL W

- prototypical progressive alignment
- similarity score with affine gap cost
- neighbor joining for tree construction
- special ‘tricks’ for gap handling
Advanced Progressive Alignment in MUSCLE

1.) alignment draft and 2.) reestimation 3.) iterative refinement

1.1 k-mer counting

k-mer distance matrix D1

1.2 UPGMA

TREE1

1.3 progressive alignment

MSA1

2.1 compute %ids from MSA1

Kimura distance matrix D2

2.2 UPGMA

TREE2

2.3 progressive alignment

MSA2

3.1 delete edge from TREE2 giving 2 subtrees

3.2 compute subtree profiles

3.3 re-align profiles

MSA

3.4 SP score better?

MSA3

No, delete

Yes, save

repeat

S. Will, 18.417, Fall 2011
Consistency-based scoring in T-Coffee

- Progressive alignment + Consistency heuristic
- Avoid mistakes when optimizing locally by modifying the scores “Library extension”
- Modified scores reflect global consistency
- Details of consistency transformation: next slide
- Merges local and global alignments
**Consistency-based scoring in T-Coffee**

**Consistency Transformation**
- For each sequence triplet: strengthen compatible edges
- This moves global information into scores
- Consistency-based scores guide pairwise alignments towards (global) consistency

### Misalignment by standard procedure

![Diagram showing misalignment](image)

### Correct alignment after library extension

![Diagram showing correct alignment](image)

### All-2-all alignments for weighting

<table>
<thead>
<tr>
<th>Sequence Pair</th>
<th>Prim. Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeqA, SeqB</td>
<td>88</td>
</tr>
<tr>
<td>SeqA, SeqC</td>
<td>77</td>
</tr>
<tr>
<td>SeqC, SeqD</td>
<td>100</td>
</tr>
<tr>
<td>SeqA, SeqD</td>
<td>100</td>
</tr>
</tbody>
</table>
Alignment Profiles

Alignment

ACGG–
ACCG–
AC–G–
TCCGG

Consensus:
ACCG–

Profile:

A : \begin{pmatrix} 0.75 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}
C : \begin{pmatrix} 0 \\ 1 \\ 0.5 \\ 0 \\ 0 \end{pmatrix}
G : \begin{pmatrix} 0 \\ 0 \\ 0.25 \\ 1 \\ 0.25 \end{pmatrix}
T : \begin{pmatrix} 0.25 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}

Remarks

• A profile of a multiple alignment consists of character frequency vectors for each column.
• The profile describes sequences of the alignment in a rigid way.
• Modeling insertions/deletions requires profile HMMs.
[The frog climbs the ladder more likely when the sun shines. Assume that the weather is hidden, but we can observe the frog.]

- **Idea:** the probability of an observation depends on a hidden state, where there are specific probabilities to change states.
- Hidden Markov Models generate observation sequences (e.g. TBTTTT) according to an (encoded) probability distribution.
- One can compute things like “most probable path given an observation sequence”, ... *(no details here)*
Profile HMMs

- Profile HMMs describe (probability distribution of) sequences in a multiple alignment (observation ≡ sequence).
- hidden states = insertion (I_i), match (M_i), deletion (D_i) in relation to consensus (state sequence ≡ alignment string)

**Alignment**

```
ACGG-
ACCG-
AC–G–
TCCGG
```

**Consensus:**

```
ACCG–
```

**Remarks**

- Profile HMMs are used to search for sequences that are similar to sequences of a given alignment (Pfam, HMMer)
- Profile HMMs can be used to construct multiple alignments
- We come back to HMMs when we discuss SCFGs.