Sequence Alignment

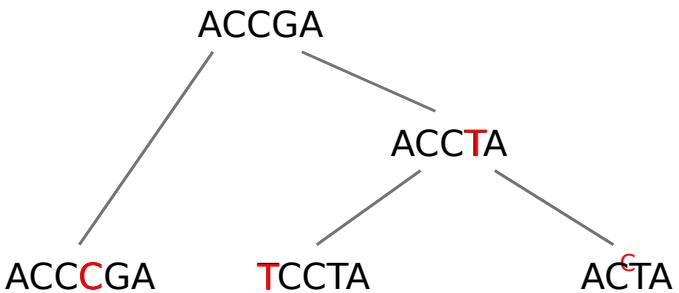
Motivation: assess similarity of sequences and learn about their evolutionary relationship

Why do we want to know this?

Example:

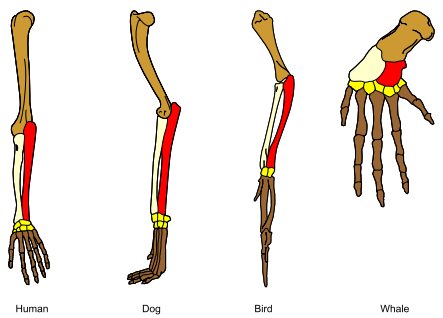
Sequences		*Alignment*
ACCCGA		ACCCGA
ACTA	$\Rightarrow_{\text{align}}$	AC--TA
TCCTA		TCC-TA

Homology: Alignment reasonable, if sequences homologous



Definition (Sequence Homology)

Two or more sequences are *homologous* iff they evolved from a common ancestor.



[Homology in anatomy]

Plan (and Some Preliminaries)

- First: study only pairwise alignment.
  Fix *alphabet* Σ, such that $- \notin \Sigma$. $-$ is called the *gap symbol*.
  The elements of Σ^* 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* Σ is a finite set (of *symbols/characters*). Σ^+ denotes the set of *non-empty words* of Σ, i.e. $\Sigma^+ := \bigcup_{i>0} \Sigma^i$. A word $x \in \Sigma^n$ has *length* n, written $|x|$. $\Sigma^* := \Sigma^+ \cup \{\epsilon\}$, where ϵ 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

ACCCGA $\rightarrow_{(C,-)}$ ACCGA $\rightarrow_{(G,T)}$ ACCTA $\rightarrow_{(-,T)}$ ATCCTA

ACCCGA $\Rightarrow_{(C,-),(G,T),(-,T)}$ ATCCTA

Recall: $- \notin \Sigma$, a, b are sequences in Σ^*

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 *cost of an edit operation* (x, y). The *cost of a sequence of edit operations* $S = e_1, \ldots, e_n$ is

$$\tilde{w}(S) = \sum_{i=1}^{n} w(e_1).$$

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 *cost of an edit operation* (x, y). The *cost of a sequence of edit operations* $S = e_1, \ldots, e_n$ is

$$\tilde{w}(S) = \sum_{i=1}^{n} w(e_1).$$

The *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 *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.

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 *cost of an edit operation* (x, y). The *cost of a sequence of edit operations* $S = e_1, \ldots, e_n$ is

$$\tilde{w}(S) = \sum_{i=1}^{n} w(e_1).$$

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

$a =$ `ACGGAT`

$b =$ `CCGCTT`

possible alignments are

$a^\diamond =$ `AC-GG-AT`
$b^\diamond =$ `-CCGCT-T`

or

$a^\diamond =$ `ACGG---AT`
$b^\diamond =$ `--CCGCT-T`

or . . . (exponentially many)

edit operations of first alignment: (A,-),(-,C),(G,C),(-,T),(A,-)

Alignment Distance

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

$$D_w(a, b) = \min\{w(a^\diamond, b^\diamond) \mid (a^\diamond, b^\diamond) \text{ is alignment of } a \text{ and } b\}.$$

Alignment Distance = Edit Distance

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
$D_w(a, b) = \min\{w(a^\diamond, b^\diamond) \mid (a^\diamond, b^\diamond) \text{ is alignment of } a \text{ and } b\}$.

Alignment Distance = Edit Distance

Theorem (Equivalence of Edit and Alignment Distance)

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

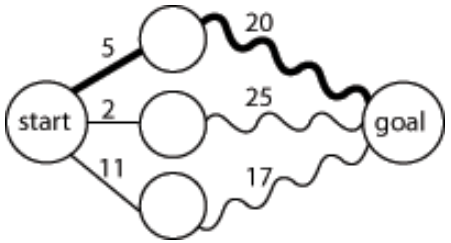
Remarks

- 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

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

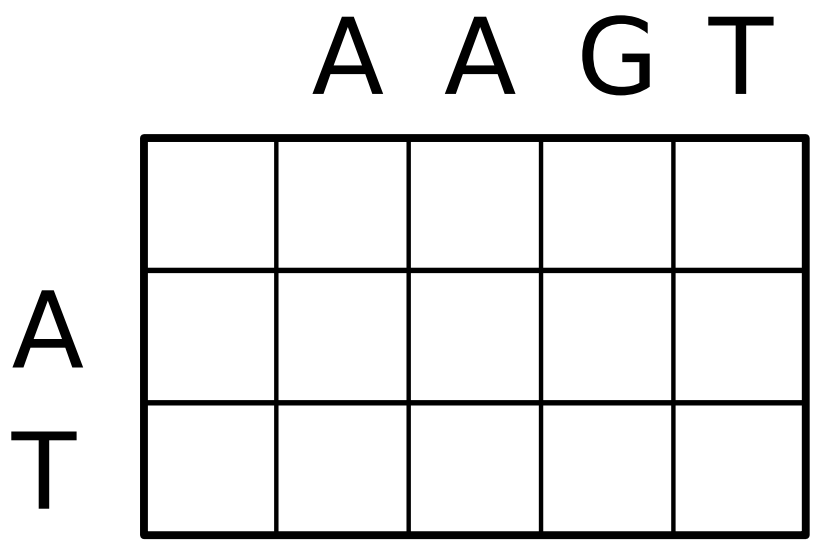
Notational remarks

- 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

- $a =$AT, $b =$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

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

		A	A	G	T
	0	1	2	3	4
A	1	0	1	2	3
T	2	1	1	2	2

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 $r = |a^\diamond|$,

$$w(a^\diamond, b^\diamond) = w(a^\diamond_{1..r-1}, b^\diamond_{1..r-1}) + w(a^\diamond_r, b^\diamond_r).$$

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* $1 \leq j \leq m$: $D_{0,j} = \sum_{k=1}^{j} w(-, b_k) = D_{0,j-1} + w(-, b_j)$
- $$D_{ij} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & (\textit{match}) \\ D_{i-1,j} + w(a_i, -) & (\textit{deletion}) \\ D_{i,j-1} + w(-, b_j) & (\textit{insertion}) \end{cases}$$

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 $r = |a^\diamond|$,

$$w(a^\diamond, b^\diamond) = w(a^\diamond_{1..r-1}, b^\diamond_{1..r-1}) + w(a^\diamond_r, b^\diamond_r).$$

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* $1 \leq j \leq m$: $D_{0,j} = \sum_{k=1}^{j} w(-, b_k) = D_{0,j-1} + w(-, b_j)$
- $$D_{ij} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & (\textit{match}) \\ D_{i-1,j} + w(a_i, -) & (\textit{deletion}) \\ D_{i,j-1} + w(-, b_j) & (\textit{insertion}) \end{cases}$$

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)

$D_{0,0} := 0$
for $i := 1$ to n **do**
  $D_{i,0} := D_{i-1,0} + w(a_i, -)$
end for
for $j := 1$ to m **do**
  $D_{0,j} := D_{0,j-1} + w(-, b_j)$
end for
for $i := 1$ to n **do**
  for $j := 1$ to m **do**

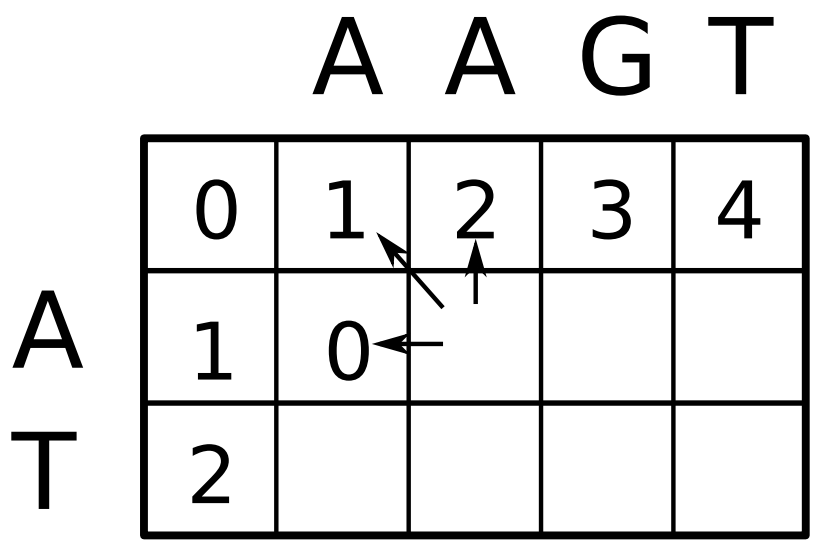
$$D_{i,j} := \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) \\ D_{i-1,j} + w(a_i, -) \\ D_{i,j-1} + w(-, b_j) \end{cases}$$

  end for
end for

Back to Example

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?

Back to Example

Example

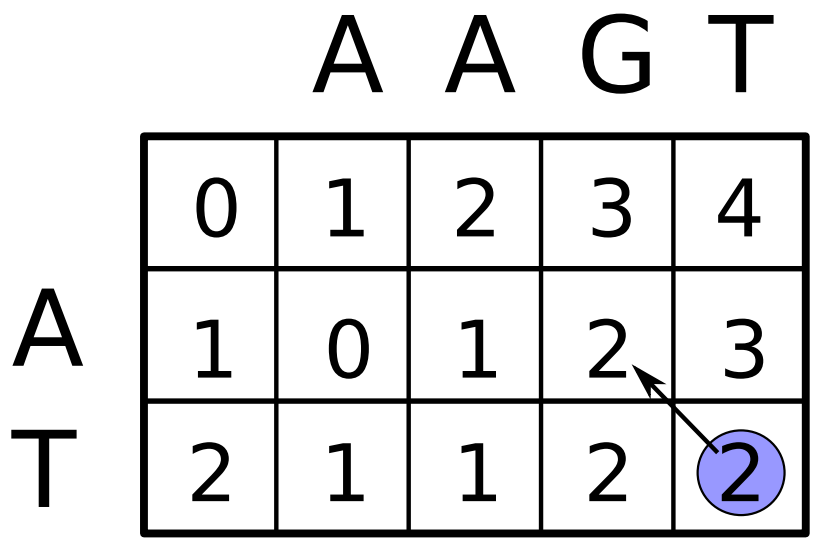
- $a =$AT, $b =$AAGT
- $w(x, y) = \begin{cases} 0 & \text{iff } x = y \\ 1 & \text{otherwise} \end{cases}$

		A	A	G	T
	0	1	2	3	4
A	1	0	1	2	3
T	2	1	1	2	2

Open: how to find best alignment?

Traceback

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

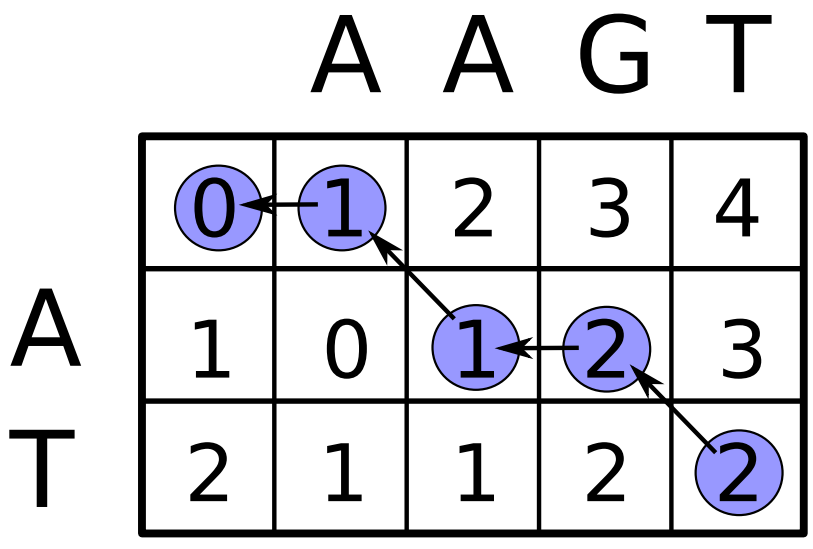


Remarks

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

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 *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)

Plan

- 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 $\begin{matrix}\texttt{GA--T}\\ \texttt{GAAGT}\end{matrix}$ and $\begin{matrix}\texttt{G-A-T}\\ \texttt{GAAGT}\end{matrix}$ 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 $g : \mathbb{N} \to \mathbb{R}$ 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

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

$$+ \sum_{x \in \Delta^{a^\diamond} \uplus \Delta^{b^\diamond}} g(|x|) \qquad (\textit{cost of gaps})$$

Example:

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

$b^\diamond =$ `-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:

- $D_{0,0} = 0$
- *for all* $1 \leq i \leq n$: $D_{i,0} = g(i)$
- *for all* $1 \leq j \leq m$: $D_{0,j} = g(j)$
- $$D_{ij} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & (\textit{match}) \\ \min_{1 \leq k \leq i} D_{i-k,j} + g(k) & (\textit{deletion of length } k) \\ \min_{1 \leq k \leq j} D_{i,j-k} + g(k) & (\textit{insertion of length } k) \end{cases}$$

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 α and β, 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 (α) and gap extension cost (β) 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 $\overset{a_i}{|}\!\!-$.
- $B_{i,j}$:= cost of best alignment of $a_{1..i}, b_{1..j}$, that ends with insertion $-\!\!\underset{b_j}{|}$.

Recursions:

$$A_{i,j} = \min \begin{cases} A_{i-1,j} + \beta & (\textit{deletion extension}) \\ D_{i-1,j} + g(1) & (\textit{deletion opening}) \end{cases}$$

$$B_{i,j} = \min \begin{cases} B_{i,j-1} + \beta & (\textit{insertion extension}) \\ D_{i,j-1} + g(1) & (\textit{insertion opening}) \end{cases}$$

$$D_{ij} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & (\textit{match}) \\ A_{i,j} & (\textit{deletion closing}) \\ B_{i,j} & (\textit{insertion closing}) \end{cases}$$

Remark: $O(n^2)$ time and space

Similarity

Definition (Similarity)

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

$$s(a^\diamond, b^\diamond) = \sum_{i=1}^{|a^\diamond|} s(a_i^\diamond, b_i^\diamond),$$

where $s : (\Sigma \cup \{-\})^2 \to \mathbb{R}$ 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 =AWGVIAC<u>AILAG</u>RS

b =VIVT<u>AIAVAG</u>YY

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

Example

a) XXXAAXXXX / YYAAYY b) XXAAAAAXXXX / YYYAAAAAYY

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

Local Alignment

Definition (Local Alignment Problem)

Let s be a similarity on alignments.

$$S_{\text{global}}(a,b) := \max_{\substack{(a^\diamond,b^\diamond)\\ \text{alignment of } a \text{ and } b}} s(a^\diamond,b^\diamond) \qquad (\textit{global similarity})$$

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

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 $(H_{i,j})_{0 \le i \le n, 0 \le j \le m}$ defined by

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

Remarks

- $S_{\text{local}}(a, b) = \max_{i,j} H_{i,j}$ (!)
- all entries $H_{i,j} \ge 0$, 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.) $\begin{array}{c|c} \dots & a_i \\ \dots & b_i \end{array}$ 2.) $\begin{array}{c|c} \dots & a_i \\ \dots & - \end{array}$ 3.) $\begin{array}{c|c} \dots & - \\ \dots & b_j \end{array}$

4.) 0, since if each of the above cases is dissimilar (i.e. negative), there is still (ϵ, ϵ).

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* $1 \leq j \leq m$: $H_{0,j} = 0$
- $$H_{ij} = \max \begin{cases} 0 & (\textit{empty alignment}) \\ H_{i-1,j-1} + s(a_i, b_j) \\ H_{i-1,j} + s(a_i, -) \\ H_{i,j-1} + s(-, b_j) \end{cases}$$

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

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

		A	C	A	A
	0	0	0	0	0
A	0	2	0	2	2
A	0	2	0	2	4
C	0	0	4	1	1

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:

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Multiple Alignment

Example: *Sequences*

$$a^{(1)} = \texttt{ACCCGAG}$$
$$a^{(2)} = \texttt{ACTACC}$$
$$a^{(3)} = \texttt{TCCTACGG}$$

$$\Rightarrow_{\text{align}} \quad A = \begin{matrix}\texttt{ACCCGA-G-}\\ \texttt{AC--TAC-C}\\ \texttt{TCC-TACGG}\end{matrix}$$

Alignment

Definition

A *multiple alignment A of K sequences* $a^{(1)}...a^{(K)}$ is a $K \times N$-matrix $(A_{i,j})_{\substack{1\le i\le K\\ 1\le j\le N}}$ (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

$$S(A) = s\begin{pmatrix} A \\ A \\ T \end{pmatrix} + s\begin{pmatrix} C \\ C \\ C \end{pmatrix} + s\begin{pmatrix} C \\ - \\ C \end{pmatrix} + s\begin{pmatrix} C \\ - \\ - \end{pmatrix} + \cdots + s\begin{pmatrix} - \\ C \\ G \end{pmatrix}$$

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

$$S(A) = \sum_{j=1}^{N} s\begin{pmatrix} A_{1j} \\ \dots \\ A_{Kj} \end{pmatrix}$$

Example

$$S(A) = s\begin{pmatrix} A \\ A \\ T \end{pmatrix} + s\begin{pmatrix} C \\ C \\ C \end{pmatrix} + s\begin{pmatrix} C \\ - \\ C \end{pmatrix} + s\begin{pmatrix} C \\ - \\ - \end{pmatrix} + \cdots + s\begin{pmatrix} - \\ C \\ G \end{pmatrix}$$

How to define $s\begin{pmatrix} x \\ y \\ z \end{pmatrix}$? as log odds $s\begin{pmatrix} x \\ y \\ z \end{pmatrix} = log\frac{Pr[x,y,z|\,\text{Related}]}{Pr[x,y,z|\,\text{Background}]}$?

Problems? Can we learn similarities for triples, 4-tuples, . . . ?

Sum-Of-Pairs Score

Idea: approximate column scores by pairwise scores

$$s\begin{pmatrix} x_1 \\ \dots \\ x_j \end{pmatrix} = \sum_{1 \le k < l \le K} s(x_k, x_l)$$

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
   $\Rightarrow$ tree, e.g. $((a^{(1)}, a^{(2)}), (a^{(3)}, a^{(4)}))$
2. align most closely related sequences first
   $\Rightarrow$ (optimally) align $a^{(1)}$ and $a^{(2)}$ by DP
3. go on $\Rightarrow$ (optimally) align $a^{(3)}$ and $a^{(4)}$ by DP
4. go on?! $\Rightarrow$ (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.

Example:

```
ACCCGA-G-
AC--TAC-C
TCC-TACGG
```

$$\equiv \binom{A}{A}\binom{C}{C}\binom{C}{-}\binom{C}{-}\binom{-}{C}\binom{-}{G}.$$

- Assign similarity to two columns from resp. A and B, e.g. $s(\binom{-}{C}, \binom{G}{C})$ by *sum-of-pairs*.

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

Consequences for progressive alignment scheme:

- Optimization only *local*.
- Commits to local decisions. "Once a gap, always a gap"

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

Align ACCG and TTGG

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

Align ACCG and TCG

		T	C	G
	0	2	4	6
A	2	3	5	7
C	4	5	3	6
C	6	7	5	6
G	8	9	8	5

Align ACCG and CTGG

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

Align TTGG and TCG

		T	C	G
	0	2	4	6
T	2	0	3	6
T	4	2	3	6
G	6	5	5	3
G	8	8	8	5

Align TTGG and CTGG

		C	T	G	G
	0	2	4	6	8
T	2	3	2	5	8
T	4	5	3	5	8
G	6	7	6	3	5
G	8	9	9	6	3

Align TCG and CTGG

		C	T	G	G
	0	2	4	6	8
T	2	3	2	5	8
C	4	2	5	5	8
G	6	5	5	5	5

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

  $\Rightarrow$ distance matrix

	$a^{(1)}$	$a^{(2)}$	$a^{(3)}$	$a^{(4)}$
$a^{(1)}$	0	9	5	8
$a^{(2)}$		0	5	3
$a^{(3)}$			0	5
$a^{(4)}$				0

  $\Rightarrow$ Cluster (e.g. UPGMA)
  $a^{(2)}$ and $a^{(4)}$ are closest, Then: $a^{(1)}$ 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 $a^{(4)}$: `TTGG` / `CTGG` , Align $a^{(1)}$ and $a^{(3)}$: `ACCG` / `-TCG`
- Align the alignments!

Align `TTGG` / `CTGG` and `ACCG` / `-TCG`

		A / -	C / T	C / C	G / G
	0	4	12	20	28
TC	8	10	...		
TT	16	⋮	⋱		
GG	24				
GG	32				

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 $a^{(4)}$: `TTGG` / `CTGG` , Align $a^{(1)}$ and $a^{(3)}$: `ACCG` / `-TCG`
- Align the alignments!

Align `TTGG`/`CTGG` and `ACCG`/`-TCG`

		A / -	C / T	C / C	G / G
	0	4	12	20	28
TC	8	10	...		
TT	16	⋮	⋱		
GG	24				
GG	32				

- $w(TC, --) =$

  $w(T,-) + w(C,-) + w(T,-) + w(C,-) = 8$
- $w(--, A-) =$

  $w(-,A) + w(-,-) + w(-,A) + w(-,-) = 4$
- $w(TC, A-) =$

  $w(T,A) + w(C,A) + w(T,-) + w(C,-) = 10$
- $w(TC, CT) =$

  $w(T,C) + w(C,C) + w(T,T) + w(C,T) = 6$
- ...

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 $a^{(4)}$: `TTGG` / `CTGG` , Align $a^{(1)}$ and $a^{(3)}$: `ACCG` / `-TCG`
- Align the alignments!

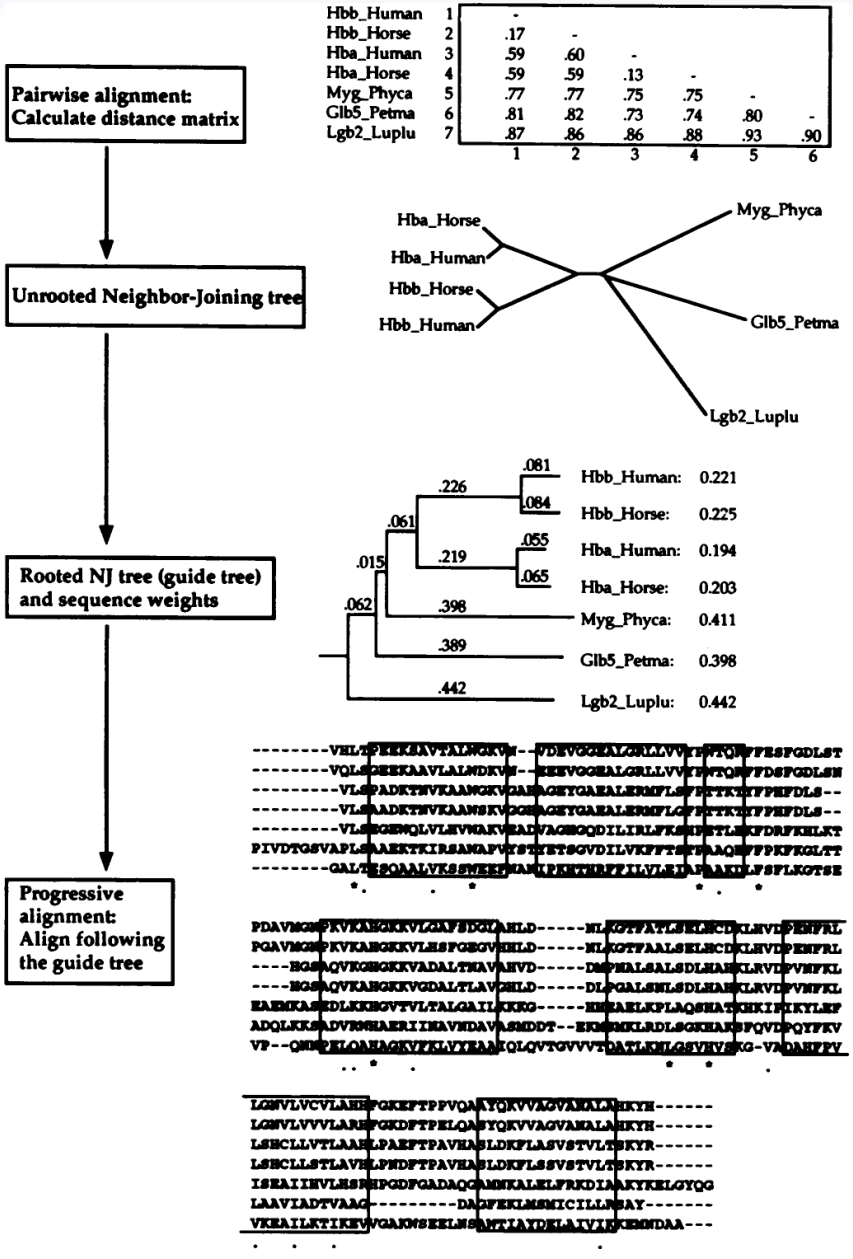
Align `TTGG` / `CTGG` and `ACCG` / `-TCG`

		A / -	C / T	C / C	G / G
	0	4	12	20	28
TC	8	10	...		
TT	16	⋮	⋱		
GG	24				
GG	32				

$\Longrightarrow$ after filling and traceback

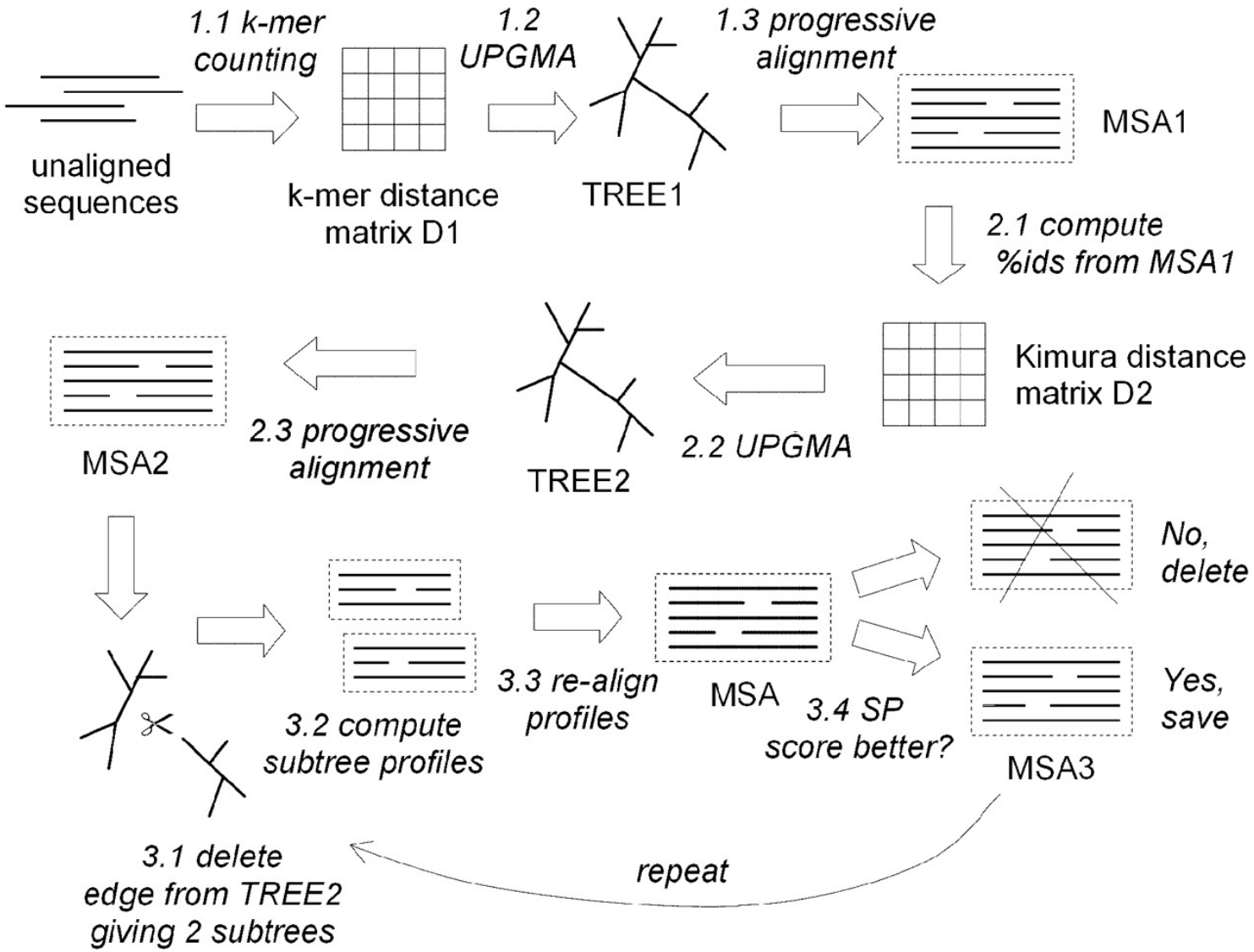
```
TTGG
CTGG
ACCG
-TCG
```

A Classical Approach: CLUSTAL W



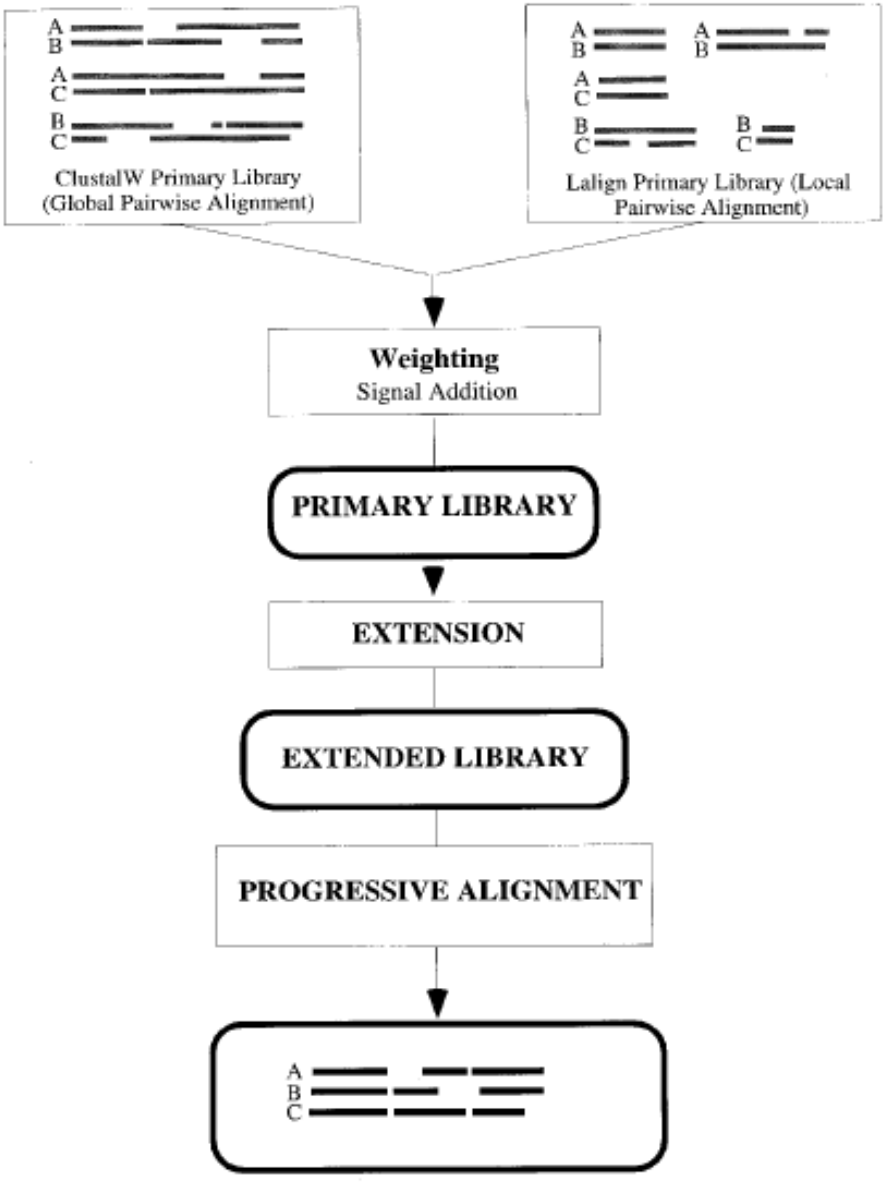
- prototypical progressive alignment
- similarity score with affine gap cost
- neighbor joining for tree construction
- special ‘tricks’ for gap handling

Advanced Progressive Alignment in MUSCLE



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

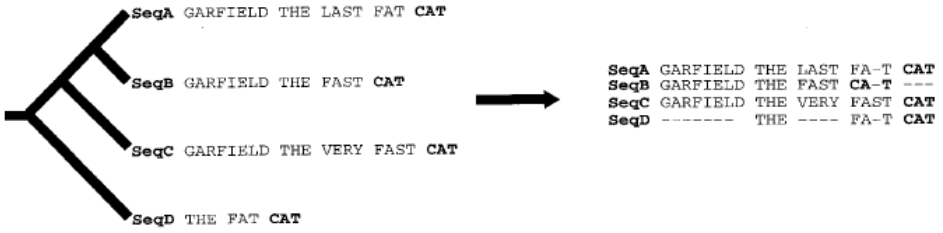
Consistency-based scoring in T-Coffee



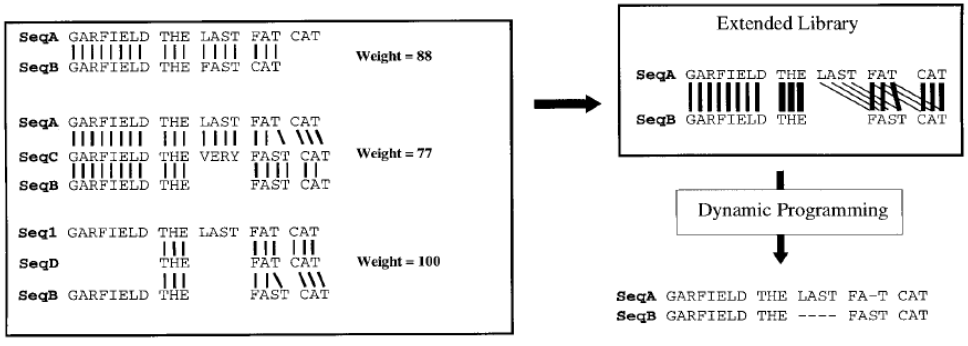
- Progressive alignment + Consistency heuristic
- Avoid mistakes when optimizing locally by modifying the scores *"Library extension"*
- Modified scores reflect global consistency
- Details of consistency transformation: next slide
- Merges local and global alignments

Consistency-based scoring in T-Coffee

Misalignment by standard procedure



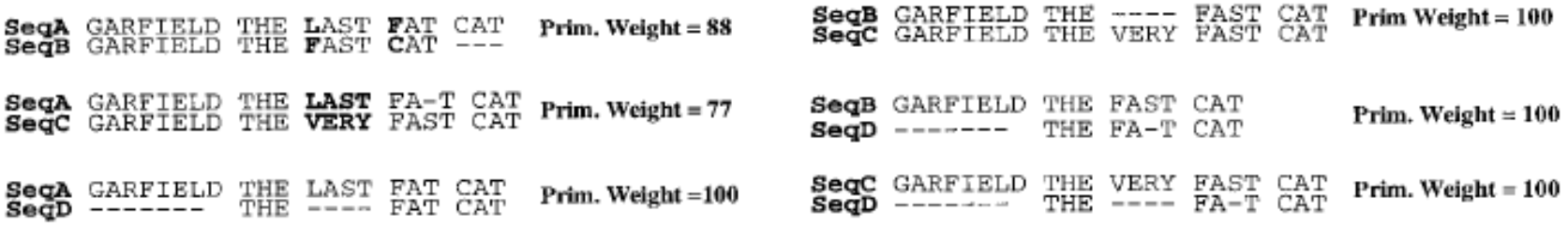
Correct alignment after library extension



Consistency Transformation

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

All-2-all alignments for weighting



Alignment Profiles

Alignment

```
ACGG-
ACCG-
AC-G-
TCCGG
```

Consensus:

```
ACCG-
```

Profile:

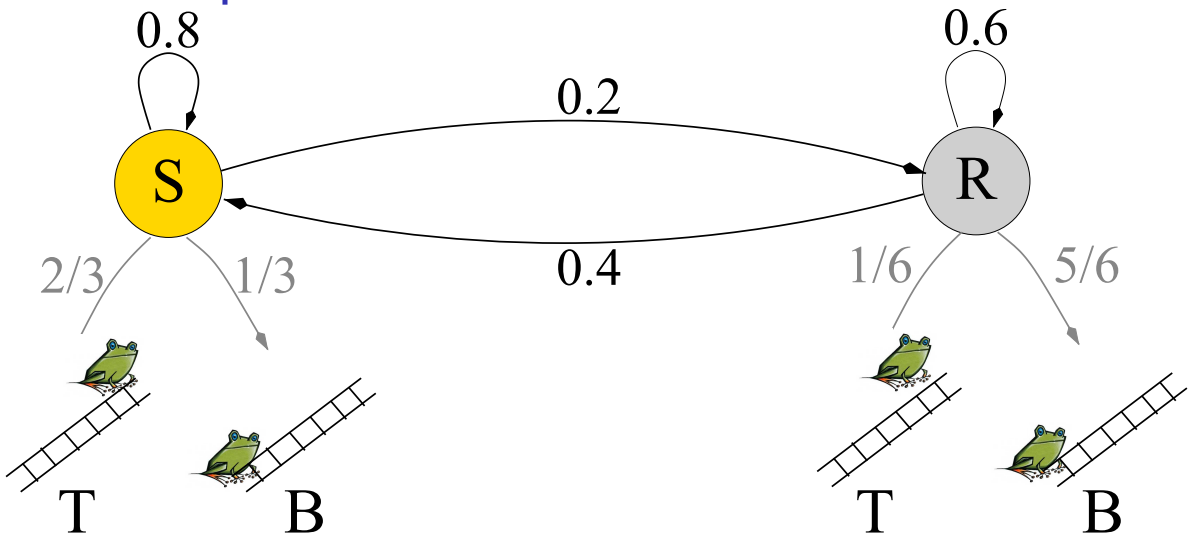
$$\begin{array}{c} A: \\ C: \\ G: \\ T: \end{array} \begin{pmatrix} 0.75 \\ 0 \\ 0 \\ 0.25 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0.5 \\ 0.25 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0.25 \\ 0 \end{pmatrix}$$

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.

Hidden Markov Models (HMMs)

Example of a simple HMM



[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. TBTTT) according to an (encoded) probability distribution.
- One can compute things like "most probable path given an observation sequence", ... *(no details here)*

Profile HMMs

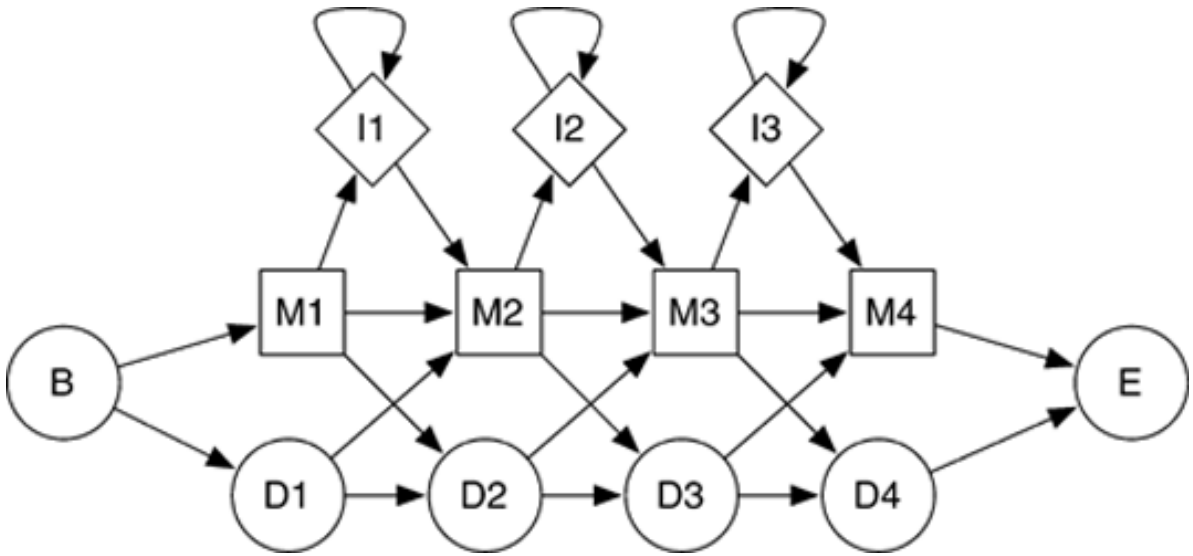
- Profile HMMs describe (probability distribution of) sequences in a multiple alignment (observation ≡ sequence).
- hidden states = insertion (I_i), match (M_i), deletion (D_i) in relation to consensus (state sequence ≡ alignment string)

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.