Bioinformatics I

03. Lecture

Dr. Dominic Rose
Bioinformatics Group, University of Freiburg

summer term 2012

Today's outline

1. Sequence Alignment
   - Motivation
   - Distances, Similarities and Cost Functions
   - Edit Distance vs. Alignment Distance
   - Needleman/Wunsch algorithm for pairwise sequence alignment

Motivation I

- Sequence alignment: search for similar sequences
- Example: schimmlig/grimmig compared to Haus/Kaffee
- Recall: search for exact patterns `TAAGTA` and `TCTACAAGTCCA`
  $\Rightarrow$ "pattern matching"
- Why do we want to search for similar (sub-)sequences:
  $\Rightarrow$ detection of homology
- Idea: homologous sequences usually have similar functions
- Infer evolutionary relationships
- Homology: orthology vs paralogy

Motivation II

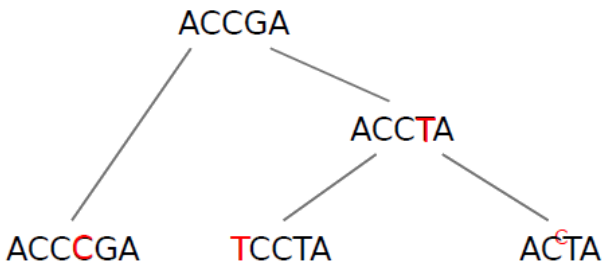
Motivation: assess similarity of sequences and learn about their evolutionary relationship

Why do we want to know this?

Example:

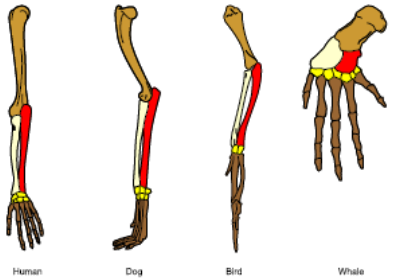
Sequences		*Alignment*
ACCCGA		ACCCGA
ACTA	$\Rightarrow_{\text{align}}$	AC--TA
TCCTA		TCC-TA

Homology: Alignment reasonable, if sequences homologous



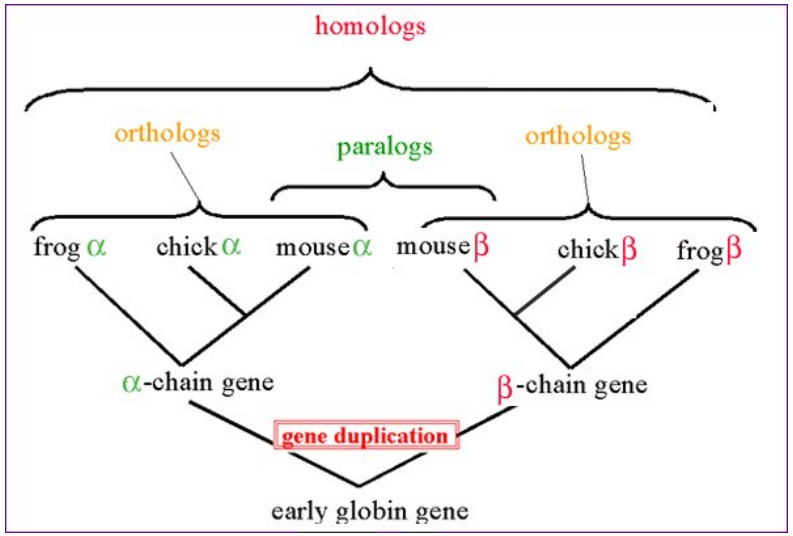
Definition (Sequence Homology)

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



[Homology in anatomy]

Homology: orthology vs paralogy



- Orthologs are genes in different species that evolved from a common ancestral gene by speciation.
- Paralogs are genes related by duplication within a genome.
- Orthologs retain the same function in the course of evolution, whereas paralogs evolve new functions.

Motivation III

- Sequence comparison: when are 2 sequences a, b with $a = a_1 a_2 a_3 ... a_n$ and $b = b_1 b_2 b_3 ... b_n$ equal?
- Algorithm?

Motivation III

- Sequence comparison: when are 2 sequences a, b with $a = a_1 a_2 a_3 ... a_n$ and $b = b_1 b_2 b_3 ... b_n$ equal?
- Algorithm?
- trivial: compare each character a_i with each b_i such that:
- $|a| = |b|$ and $\forall i \in 1..n : a_i = b_i$

Motivation III

- Sequence comparison: when are 2 sequences a, b with $a = a_1 a_2 a_3 ... a_n$ and $b = b_1 b_2 b_3 ... b_n$ equal?
- Algorithm?
- trivial: compare each character a_i with each b_i such that:
- $|a| = |b|$ and $\forall i \in 1..n : a_i = b_i$
- However, a qualitative distance/similarity measure for pairwise string comparisons is desirable
  (e.g., what if a, b have different lengths?)
- similar sequences $\rightarrow$ small distance (high similarity)

Motivation IV

- Search for similar sequences ⇒ “probably” homologous
- Hence: sequence alignment (search for similarity) must be based on a model of evolution
- two views on sequence alignment:
  - biologically: sequence of events to transform one sequence into another one (sequence of events ⇒ edit operations)

```
schimmlig
   ||| ||
 grimm ig
```

  - technically: align sequences to “see” similarities

Outline and 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 their equivalence
  - define alignment problem and introduce efficient algorithm to solve global pairwise alignment
  - gap penalties, other variants: local, semi-global alignment, ...
- Later: extend pairwise alignment to multiple alignment

Outline and 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 their equivalence
  - define alignment problem and introduce efficient algorithm to solve global pairwise alignment
  - gap penalties, other variants: local, semi-global 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 as $|x|$.
$\Sigma^* := \Sigma^+ \cup \epsilon$, where ϵ denotes the empty word of length 0.

Distances vs. Similarities

- Two ways to quantify "similarity" of two sequences:
  Distance and Similarity, which are (somehow) dual:
- Both are (cost-)functions that associate a numeric value with a pair of sequences
  - Idea similarity measure:
    higher values indicate greater similarity
  - Idea distance measure:
    the larger the distance, the smaller the similarity (and vice versa).
- Distance measures satisfy the mathematical axioms of a metric (sequences are treated as points in a metric space).
  In particular, distance values are never negative.
- In most cases, distance and similarity measures are interchangable (small distance $\iff$ high similarity).

Cost functions: Hamming distance

- Maybe the simplest notion of distance (1950, Richard Hamming)
- For two sequences of equal length, we count the character positions in which they differ.
- Example: ”toned” vs ”roses” → 3.
- Measures the minimum number of substitutions required to change one string into the other, or the number of errors that transform one string into the other.
- More examples:

```
sequence s          AAT   AGCAA   AGCACACA
sequence t          TAA   ACATA   ACACACTA
HammingDist(s,t)     2      3         6
```

- Sometimes useful, but not very flexible.
  - Sequences may have different length.
  - No fixed correspondence between character positions.

Cost functions: Hamming distance

Further limitations:

- In the mechanism of DNA replication, errors like deleting or inserting a nucleotide are not unusual.
- Although the rest of the sequences might be identical, such position-wise shifts lead to exaggerated values in the Hamming distance.
- Example:

```
sequence s          AGCACACA
sequence t          ACACACTA
HammingDist(s,t)        6
```

- According to the Hamming distance *s* and *t* are apart by 6 characters (out of 8).
- On the other hand, by deleting *G* from *s* and the *T* from t, both become equal to ACACACA. In this sense, they are only two characters apart!

Edit Operations (informal)

- Idea: model the distance of s and t by considering the simple, one-character edit operations that turn s into t.
- We introduce a gap character "-" and say that the pair
  - (a, a) denotes a match (no change from s to t),
  - (a, b) denotes a substitution of a (in s) by b (in t), where $a \neq b$,
  - $(a, -)$ denotes a deletion of character a (in s),
  - $(-, b)$ denotes an insertion of character b (in t),
- Note: the problem is symmetric in s and t
  (deletion in s can be seen as an insertion in t and vice versa).
- The alignment of s and t is an arrangement by position, where s and t can be padded with gap symbols to achieve same lengths.
- Example:

```
s: AGCACAC-A     or    AG-CACACA
t: A-CACACTA           ACACACT-A
```

Edit Operations (informal)

- Example:

```
   s: AGCACAC-A      or      AG-CACACA
   t: A-CACACTA              ACACACT-A

Left: Match  (A,A)   Right: Match   (A,A)
      Delete(G,-)           Replace(G,C)
      Match  (C,C)          Insert  (-,A)
      Match  (A,A)          Match   (C,C)
      Match  (C,C)          Match   (A,A)
      Match  (A,A)          Match   (C,C)
      Match  (C,C)          Replace(A,T)
      Insert(-,T)           Delete  (C,-)
      Match  (A,A)          Match   (A,A)
```

Edit Operations vs. distance measures

- The edit protocol can be turned into a measure of distance by assigning a "cost" or "weight" to each operation.
- For example, for arbitrary characters x, y from the alignment A, we may define

$$\delta(a,b) = \begin{Bmatrix} 0 & a = b & \textit{Match} \\ 1 & a \neq b & \textit{Mismatch} \end{Bmatrix}$$
$$\delta(a,\_) = 1 \quad \textit{Deletion}$$
$$\delta(\_,b) = 1 \quad \textit{Insertion}$$

also known as unit cost model or Levenshtein distance.

Cost functions: Levenshtein distance

- 1965, named after Vladimir Levenshtein
- The Levenshtein Distance between two words/sequences is the minimal number of substitutions, insertions and deletions to transform one into the other.

Cost functions: Levenshtein distance

- 1965, named after Vladimir Levenshtein
- The Levenshtein Distance between two words/sequences is the minimal number of substitutions, insertions and deletions to transform one into the other.
- Corresponds to the "edit distance"
  - Number of edit operations $|O|$ for a sequence of edit operations O (allowed edit operations: insertion, deletion, and substitution)
  - Application of the unit cost model to the previous example of the alignment of s, and t, we obtain the following distances:

```
     s :  AGCACAC–A        or        AG–CACACA
     t :  A–CACACTA                  ACACACT–A
 costs :        2                        4
```

  - Obviously, the left-hand assignment is optimal under the unit cost model, and hence the edit distance is 2.

Cost functions: Levenshtein distance

- 1965, named after Vladimir Levenshtein
- The Levenshtein Distance between two words/sequences is the minimal number of substitutions, insertions and deletions to transform one into the other.
- Corresponds to the "edit distance"
  - Number of edit operations $|O|$ for a sequence of edit operations O (allowed edit operations: insertion, deletion, and substitution)
  - Application of the unit cost model to the previous example of the alignment of s, and t, we obtain the following distances:

```
     s :  AGCACAC–A        or        AG–CACACA
     t :  A–CACACTA                  ACACACT–A
 costs :        2                        4
```

  - Obviously, the left-hand assignment is optimal under the unit cost model, and hence the edit distance is 2.
- In biology, operations have different cost. (Why?)

Cost functions: Levenshtein distance

- 1965, named after Vladimir Levenshtein
- The Levenshtein Distance between two words/sequences is the minimal number of substitutions, insertions and deletions to transform one into the other.
- Corresponds to the "edit distance"
  - Number of edit operations $|O|$ for a sequence of edit operations O (allowed edit operations: insertion, deletion, and substitution)
  - Application of the unit cost model to the previous example of the alignment of s, and t, we obtain the following distances:

```
     s :  AGCACAC–A        or        AG–CACACA
     t :  A–CACACTA                  ACACACT–A
 costs :        2                        4
```

  - Obviously, the left-hand assignment is optimal under the unit cost model, and hence the edit distance is 2.
- In biology, operations have different cost. (Why?)

  (replacing an amino acid by a biochemically similar one should weight less than a replacement by an amino acid with totally different properties)

Edit Operations (formal)

Definition

Given an alphabet Σ (finite) with $- \notin \Sigma$ (gap symbol).
An edit operation is a pair $(x, y) \in (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\})$.
A pair (x, y) is called

- substitution, if $x \neq -$ and $y \neq -$
- insertion, if $x = -$ and $y \neq -$
- deletion, if $x \neq -$ and $y = -$

We write $a \rightarrow_{(x,y)} b$ if b is generated from a by the replacement of x with y (if (x, y) substitution) or by deletion of one x (if (x, y) deletion) or respectively insertion of one y (if (x, y) insertion).

- Examples:

deletion:	ATTACG	$\rightarrow_{(G,-)}$	ATTAC
substitution:	ACCA	$\rightarrow_{(A,T)}$	ACCT
insertion:	ATAT	$\rightarrow_{(-,A)}$	ATATA

Sequence of Edit Operations

Definition

Let $O = o_1 \dots o_n$ be a sequence of edit operations, and let

$$a \Rightarrow_O b \leftrightarrow a = a^{(0)} \rightarrow_{o_1} a^{(1)} \rightarrow_{o_2} \dots \rightarrow_{o_n} a^{(n)} = b$$

be a sequence of words such that $a^{(i-1)} \rightarrow_{o_i} a^{(i)}$.
Then we write $a \Rightarrow_O b$.

Sequence of Edit Operations

Definition

Let $O = o_1 \dots o_n$ be a sequence of edit operations, and let

$$a \Rightarrow_O b \leftrightarrow a = a^{(0)} \rightarrow_{o_1} a^{(1)} \rightarrow_{o_2} \dots \rightarrow_{o_n} a^{(n)} = b$$

be a sequence of words such that $a^{(i-1)} \rightarrow_{o_i} a^{(i)}$.
Then we write $a \Rightarrow_O b$.

- Example:

$$\text{schimmlig} \overset{4\ \textit{deletions}}{\Rightarrow} \text{immig} \overset{2\ \textit{insertions}}{\Rightarrow} \text{grimmig:} \qquad \text{distance } 6$$
$$\text{Haus} \overset{3\ \textit{deletions}}{\Rightarrow} \text{a} \overset{5\ \textit{insertions}}{\Rightarrow} \text{Kaffee:} \qquad \text{distance } 8$$

- needed:
  "minimal" sequence O for given (a, b) transforming $a \Rightarrow_O b$
  $\Rightarrow$ cost function

Edit Distance: Cost and Problem Definition

Definition

Let $w : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \rightarrow \mathbb{R}$, such that $w(x, y)$ is the cost of an edit operation (x, y). w is called a cost function.
The cost of a sequence of edit operations $O = o_1 ... o_n$ is defined as

$$w(O) = \sum_{i=1}^{n} w(o_i)$$

The edit distance of sequences $a, b \in \Sigma^*$ is defined as

$$d_w(a, b) = min\{w(O) | a \Rightarrow_O b\}$$

("Needleman-Wunsch cost function")

Edit Distance: Cost and Problem Definition

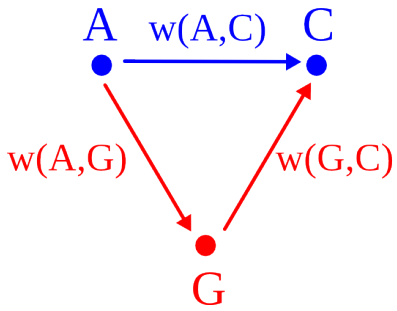
Remarks

- Natural 'evolution-motivated' problem definition
- ∞-many edit operations transforming word a into word b
- Not obvious how to compute edit distance efficiently
- Which kind of cost function should be allowed?

$\rightarrow$ Define alignment distance (later)

Metric

- The triangle inequality should be satisfied:



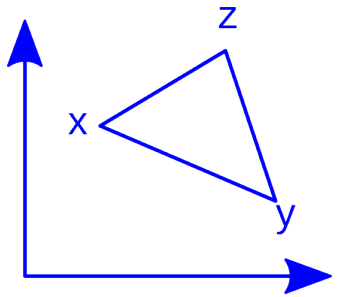
$w(A,C) \leqslant w(A,G) + w(G,C)$

Definition

A cost function w is called a metric if

1. $w(x, y) \geq 0$ and $w(x, y) = 0 \leftrightarrow x = y$ (positive definiteness)
2. $w(x, y) = w(y, x)$ (symmetry)
3. $w(x, z) \leq w(x, y) + w(y, z)$ (subadditivity, triangle inequality)

Example: Euclidean Distance



Example of a cost function: unit costs

Unit costs:

$$\delta(a,b) = \begin{cases} 0 & a = b \quad \textit{Match} \\ 1 & a \neq b \quad \textit{Mismatch} \end{cases}$$

$$\delta(a,\_) = 1 \quad \textit{Deletion}$$

$$\delta(\_,b) = 1 \quad \textit{Insertion}$$

Remarks:

- Equivalent use of “score” and “distance” (costs, weights, . . .)
- Alignment-scores have to be maximized (in order to get an optimal alignment); measure sequence similarities.
- Alignment-distances have to be minimized; measure sequence differences.
- Using the above cost function, do we have to maximize or minimize?

Pairwise alignment, example

The alignment problem is an optimization problem, example:

- Given: Two sequences *s* and *t*; score function: unit costs.
- Hypothesis: *s* and *t* have common ancestors (are homolog).
- Question: Which positions in *s* and *t* are homolog?

s=GAC and t=GC can be aligned in many ways:

Möglichkeit	Alignment	Score
1	GAC GC-	0+1+1=2
2	GAC-- ---GC	1+1+1+1+1=5
3	GAC G-C	0+1+0=1

Definition Alignment

Definition

Given two words $a, b \in \Sigma^*$.
An alignment of (a, b) consists of two sequences $a', b' \in (\Sigma \cup \{-\})^*$ such that

1. $|a'| = |b'|$
   (alignment strings have the same lengths)
2. $\forall 1 \leq i \leq |a'| : \neg(a'_i = - = b'_i)$
   (aligning a gap with a gap is forbidden)
3. $a'|_\Sigma = a$ und $b'|_\Sigma = b$
   (removing gaps from the alignment string yields the original sequences)

Definition Alignment

Definition

Given two words $a, b \in \Sigma^*$.
An alignment of (a, b) consists of two sequences $a', b' \in (\Sigma \cup \{-\})^*$ such that

1. $|a'| = |b'|$
   (alignment strings have the same lengths)
2. $\forall 1 \leq i \leq |a'| : \neg(a'_i = - = b'_i)$
   (aligning a gap with a gap is forbidden)
3. $a'|_\Sigma = a$ und $b'|_\Sigma = b$
   (removing gaps from the alignment string yields the original sequences)

- Alignment: "adjust sequences by insertion of gaps"
- How to do it in an "optimal" way?
- Example: a=ACGGAT, b=CCGCTT

possible alignments are

$$\begin{array}{ll} a' = \texttt{AC-GG-AT} \\ b' = \texttt{-CCGCT-T} \end{array} \quad \text{or} \quad \begin{array}{ll} a' = \texttt{ACGG---AT} \\ b' = \texttt{--CCGCT-T} \end{array} \quad \text{or} \ldots \text{(exponentially many)}$$

Alignment Distance

Definition

Given an alignment (a', b') of (a, b).
The cost of the alignment given a cost function w on edit operations is

$$w(a', b') = \sum_{i=1}^{|a'|} w(a'_i, b'_i)$$

The alignment distance of two words $a, b \in \Sigma^*$ is

$$D_w(a, b) = \min \{ w(a', b') \mid (a', b') \text{ is alignment of } (a, b) \}$$

Naïve solution

- We only have to analyze a finite number of alignments
- Naïve solution: generate all of them, keep the best one
- **But:**
  - Number of aln. grows exponentially with the sequence lengths
  - Naïve approach (enumerating all alignments) not feasible
  - There are

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{\pi n}}$$

    possible (global) alignments for 2 sequences of length n.
  - Example: 2 sequences of length 100 can form approx. 10^{77} possible alignments
- Trick: reduce edit distance to alignment distance
- We will see: the alignment distance is computed effciently by dynamic programming (using Bellman's Principle of Optimality).

Edit Distance = Alignment 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

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

Definition (Alignment Distance)

The alignment distance of a and b is

$$D_w(a, b) = \min\{w(a', b') \mid (a', b') \text{ is alignment of } (a, b)\}$$

Edit Distance = Alignment Distance

Theorem (Equivalence of Edit and Alignment Distance)

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

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, proof e.g. via induction on length of sequence of edit ops O)

Additivity

- Property of alignment distance: **additivity**
- Example: Levenshtein distance

$$\begin{array}{rcc|ccc} a' = & \text{A} & \text{C} & \text{-} & \text{G} & \text{T} \\ b' = & \text{-} & \text{C} & \text{A} & \text{G} & \text{T} \end{array}$$

- Split it, treat it as two separate alignments

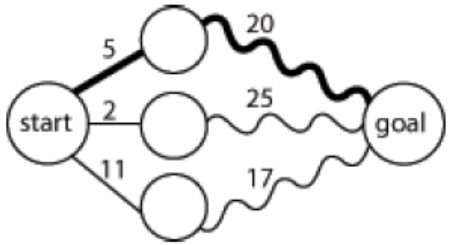
$$\begin{array}{rcc} c' = & \text{A} & \text{C} \\ d' = & \text{-} & \text{C} \\ & 1 & \end{array} \qquad \begin{array}{rccc} e' = & \text{-} & \text{G} & \text{T} \\ f' = & \text{A} & \text{G} & \text{T} \\ & & 1 & \end{array}$$

- $$\begin{aligned} w(a', b') = 2 \quad &= w(c'e', d'f') \\ &= w(c', d') + w(e', f') \end{aligned}$$
- Additivity, basis to solve the pairwise alignment problem by Dynamic Programming.

Dynamic Programming (DP)

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

Dynamic Programming, further remarks

- Divide and Conquer strategy (algorithm design pattern):
  - Break the problem into smaller sub-problems
  - Solve the smaller problems optimally
  - Use the sub-problem solutions to construct an optimal solution for the original problem.
- Solve each sub-problem only once, thus reduce the number of computations.
- Top-down and bottom-up phase

DP & Sequence Alignment

- Idea:
  - Split problem into sub-problems
  - Optimal solution can be obtained by combination of “right” sub-problems
  - Avoid recomputation of same sub-problem $\Rightarrow$ tabularization $\Rightarrow$ store in a matrix
- Hence: we need to define the sub-problems
  - Prefix alignments
  - That are optimal aligments for $a_1 \ldots a_i$ with $b_1 \ldots b_j$
  - Alignment matrix (Needleman/Wunsch matrix)

Alignment matrix (for prefix alignments)

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 $D := (D_{ij})_{0 \le i \le n, 0 \le j \le m}$ defined by

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

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 $x > y$.

Needleman/Wunsch: Example

- example: $a = AT$ and $b = AAGT$.
- cost function: levensthein distance $w(x, y) = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{else} \end{cases}$

$(D_{i,j}) =$

		A	A	G	T
	$\boxed{0}$	1	2	3	4
A	1	0	1	2	3
T	2	1	1	2	$\boxed{2}$

$D_{0,0}$: opt. alignment of $\epsilon = a_1 \ldots a_0$ with $\epsilon = b_1 \ldots b_0$

opt. alignment of A and AA

opt. alignment of A with AAG, best alignments:

$$\begin{matrix} - & \mathrm{A} & - \\ \mathrm{A} & \mathrm{A} & \mathrm{G} \end{matrix} \quad \text{and} \quad \begin{matrix} \mathrm{A} & - & - \\ \mathrm{A} & \mathrm{A} & \mathrm{G} \end{matrix}$$

$$\begin{matrix} \mathrm{A} & - & - & \mathrm{T} \\ \mathrm{A} & \mathrm{A} & \mathrm{G} & \mathrm{T} \end{matrix}$$

- **Remark:** cost for optimal aligment of a, b can be found in $\boxed{} = D_{|a|,|b|}$

“Alignment decomposition”

- Split of alignments, example:

$$w\left(\begin{array}{cccccccc} A & C & - & G & G & - & A & T \\ - & C & C & G & C & T & - & T \end{array}\right) = w\left(\begin{array}{ccccccc} A & C & - & G & G & - & A \\ - & C & C & G & C & T & - \end{array}\right) + w\begin{pmatrix} T \\ T \end{pmatrix}$$

Proposition

Let (u', v') be some alignment which is optimal for prefixes of $a_1 \dots a_i$ and $b_1 \dots b_j$. Then

1. $$\begin{aligned} w(u', v') &= w(u'_1 \dots u'_r, v'_1 \dots v'_r) \\ &= w(u'_1 \dots u'_{r-1}, v'_1 \dots v'_{r-1}) + w(u'_r, v'_r) \end{aligned}$$
2. $w(u'_1 \dots u'_r, v'_1 \dots v'_r)$ must be optimal as well

“Alignment decomposition”

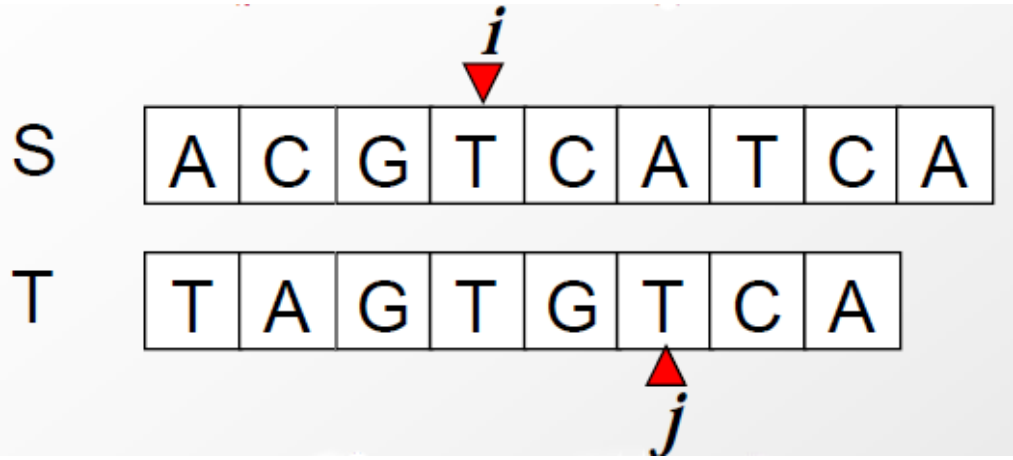
- How does an alignment column basically look like?
  There are three possible cases:
  1.) Substitution: (a_i, b_j)
  2.) Insertion (in a): $(a_i, -)$
  3.) Deletion (in a): $(-, b_j)$

“Alignment decomposition”

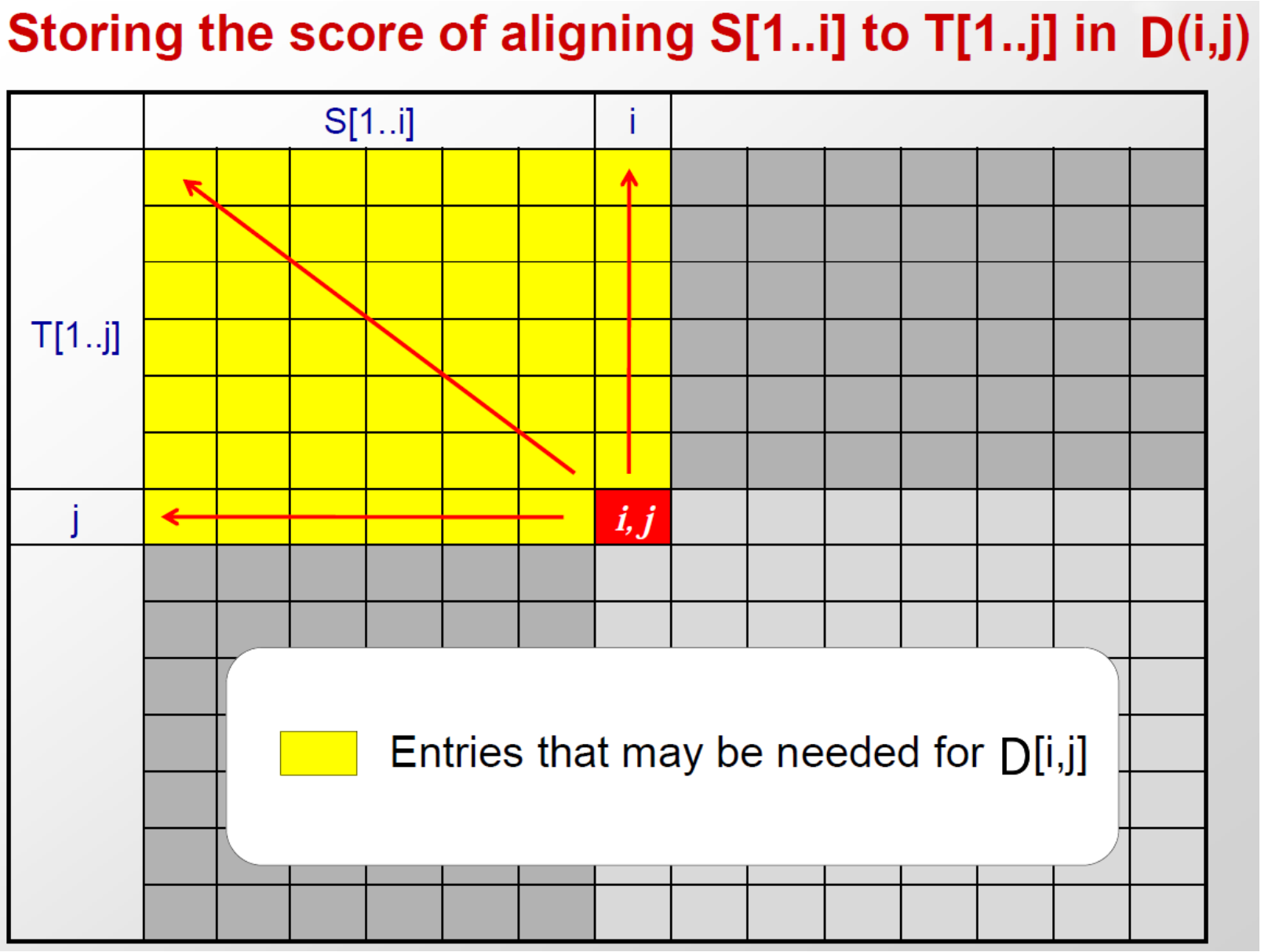
- How does an alignment column basically look like?
  There are three possible cases:
  1.) Substitution: (a_i, b_j)
  2.) Insertion (in a): $(a_i, -)$
  3.) Deletion (in a): $(-, b_j)$
- Assume, the optimal alignment of $a_1 \dots a_{i-1}$ and $b_1 \dots b_{j-1}$ having costs $D_{i-1,j-1}$ is already known.
- Then again, three cases for next column (a_i, b_j).

1.)	Substitution:	$D_{i-1,j-1}$	$+$	$w(a_i, b_j)$	$= D_{i,j}$
2.)	Insertion (in a):	$D_{i-1,j}$	$+$	$w(a_i, -)$	$= D_{i,j}$
3.)	Deletion (in a):	$D_{i,j-1}$	$+$	$w(-, b_j)$	$= D_{i,j}$

Example

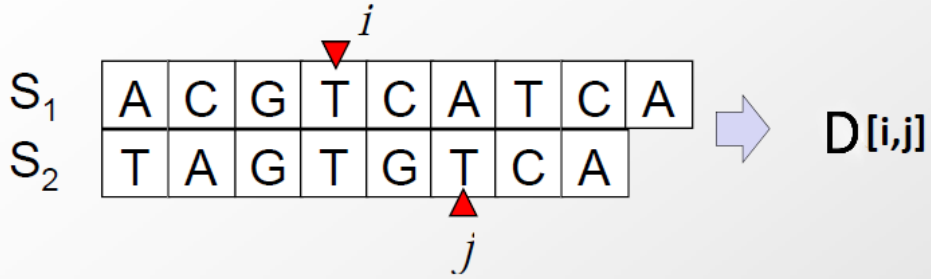


Example


Storing the score of aligning S[1..i] to T[1..j] in D(i,j)
S[1..i]
i
T[1..j]
j
i, j
Entries that may be needed for D[i,j]

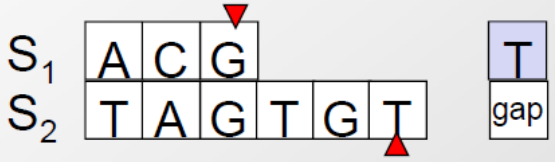
Example

Reusing computation: recursion formula



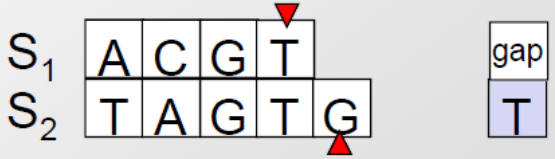
D[i,j]

- Score of best alignment of $S_1[1..i]$ and $S_2[1..j]$ is max of:
  - Score of S[1..i-1],T[1..j] + cost of gap in S



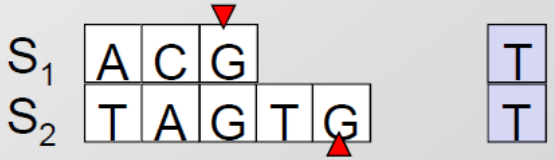
D[i,j]=D[i-1,j]+gap

  - Score of S[1..i], T[1..j-1] + cost of gap in T



D[i,j]=D[i,j-1]+gap

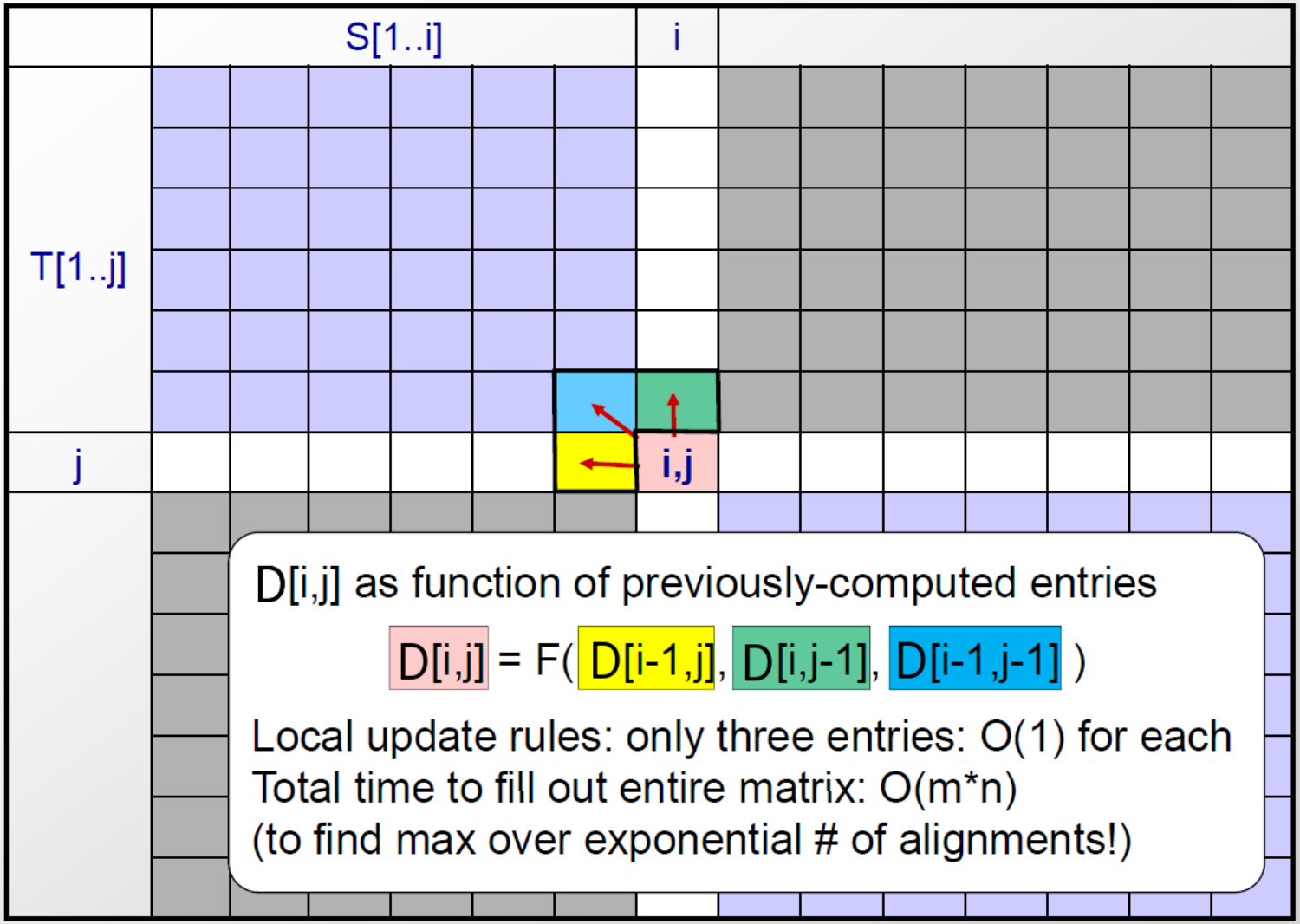
  - Score of S[1..i-1], T[1..j-1] + match cost of T[i] S[j] chars



D[i,j]=D[i-1,j-1]+cost('T','T')

Example

(1, 2, 3) Store score of aligning (i,j) in matrix D(i,j)



D[i,j] as function of previously-computed entries

D[i,j] = F(D[i-1,j], D[i,j-1], D[i-1,j-1])

Local update rules: only three entries: O(1) for each
Total time to fill out entire matrix: O(m*n)
(to find max over exponential # of alignments!)

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

Needleman/Wunsch recursion scheme

Theorem (Needleman/Wunsch)

recursion for $D_{i,j}$*:* $\forall i, j > 0 \; D_{i,j} = \min \left\{ \begin{array}{l} D_{i-1,j-1} + w(a_i, b_j), \\ D_{i-1,j} + w(a_i, -), \\ D_{i,j-1} + w(-, b_j) \end{array} \right\}$

initialization:

$$\begin{aligned}
D_{0,0} &= \textit{opt. alignment of } (\epsilon, \epsilon) && = \quad 0 \\
D_{0,j} &= \textit{opt. alignment of } (\epsilon, b_1 \ldots b_j) \\
&= w \begin{pmatrix} - & - & - & \ldots & - & - & - \\ b_1 & b_2 & b_3 & \ldots & b_{j-2} & b_{j-1} & b_j \end{pmatrix} \\
&= w \begin{pmatrix} - \\ b_1 \end{pmatrix} + w \begin{pmatrix} - \\ b_2 \end{pmatrix} + \ldots + w \begin{pmatrix} - \\ b_j \end{pmatrix} && = \quad \textstyle\sum_{k=1}^{j} w(-, b_k) \\
D_{i,0} &= \textit{opt. alignment of } (a_1 \ldots a_j, \epsilon) && = \quad \textstyle\sum_{k=1}^{i} w(a_k, -)
\end{aligned}$$

Complexity of Needleman/Wunsch

- $|a| = m$, $|b| = n \Rightarrow (m+1) \cdot (n+1)$ entries.
  $\Rightarrow O(m * n)$ many entries.
- cost for one entry: $O(1)$
  $\Rightarrow$ **total:** $O(m * n)$ time and space

Traceback example

- **needed:** optimal alignment $\Rightarrow$ *traceback*
- **example:**

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

resulting alignments:

Traceback starts here

For $D_{2,4}$ check three directions:

- $\uparrow$ $D_{2,4} \neq D_{1,4} + w(-, T) = 4$ ↯

Traceback example

- **needed:** optimal alignment $\Rightarrow$ *traceback*
- **example:**

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

resulting alignments:

Traceback starts here

For $D_{2,4}$ check three directions:

- $\uparrow$ $D_{2,4} \neq D_{1,4} + w(-,T) = 4$
- $\leftarrow$ $D_{2,4} \neq D_{2,3} + w(T,-) = 3$

Traceback example

- **needed:** optimal alignment $\Rightarrow$ *traceback*
- **example:**

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

resulting alignments:

Traceback starts here

For $D_{2,4}$ check three directions:

- $\uparrow$ $D_{2,4} \neq D_{1,4} + w(-,T) = 4$
- $\leftarrow$ $D_{2,4} \neq D_{2,3} + w(T,-) = 3$
- $\nwarrow$ $D_{2,4} = D_{1,3} + w(T,T) = 2$

Traceback example

- **needed:** optimal alignment $\Rightarrow$ *traceback*
- **example:**

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

resulting alignments:

A – – T
A A G T

Traceback starts here

For $D_{2,4}$ check three directions:

- $D_{2,4} \neq D_{1,4} + w(-,T) = 4$
- $D_{2,4} \neq D_{2,3} + w(T,-) = 3$
- $D_{2,4} = D_{1,3} + w(T,T) = 2$

Traceback example

- **needed:** optimal alignment $\Rightarrow$ *traceback*
- **example:**

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

resulting alignments:

$$\begin{matrix} A & - & - & T \\ A & A & G & T \end{matrix} \quad \text{and} \quad \begin{matrix} - & A & - & T \\ A & A & G & T \end{matrix}$$

Traceback starts here

For $D_{2,4}$ check three directions:

- ↑ $D_{2,4} \neq D_{1,4} + w(-,T) = 4$
- ← $D_{2,4} \neq D_{2,3} + w(T,-) = 3$
- ↖ $D_{2,4} = D_{1,3} + w(T,T) = 2$

Traceback example

- example:

		A	A	G	T
	$\emptyset$	$\{\leftarrow\}$	$\{\leftarrow\}$	$\{\leftarrow\}$	$\{\leftarrow\}$
A	$\{\uparrow\}$	$\{\nwarrow\}$	$\{\leftarrow, \nwarrow\}$	$\{\leftarrow\}$	$\{\leftarrow\}$
T	$\{\uparrow\}$	$\{\uparrow\}$	$\{\leftarrow\}$	$\{\leftarrow, \nwarrow\}$	$\{\nwarrow\}$

- **traceback:** sequence of arrows, starting from right/bottom and ending at top/left, following the directions
- **here:** 2 possible tracebacks

$\nwarrow \leftarrow \leftarrow \nwarrow$ $\qquad$ $\leftarrow \nwarrow \leftarrow \nwarrow$

Traceback example

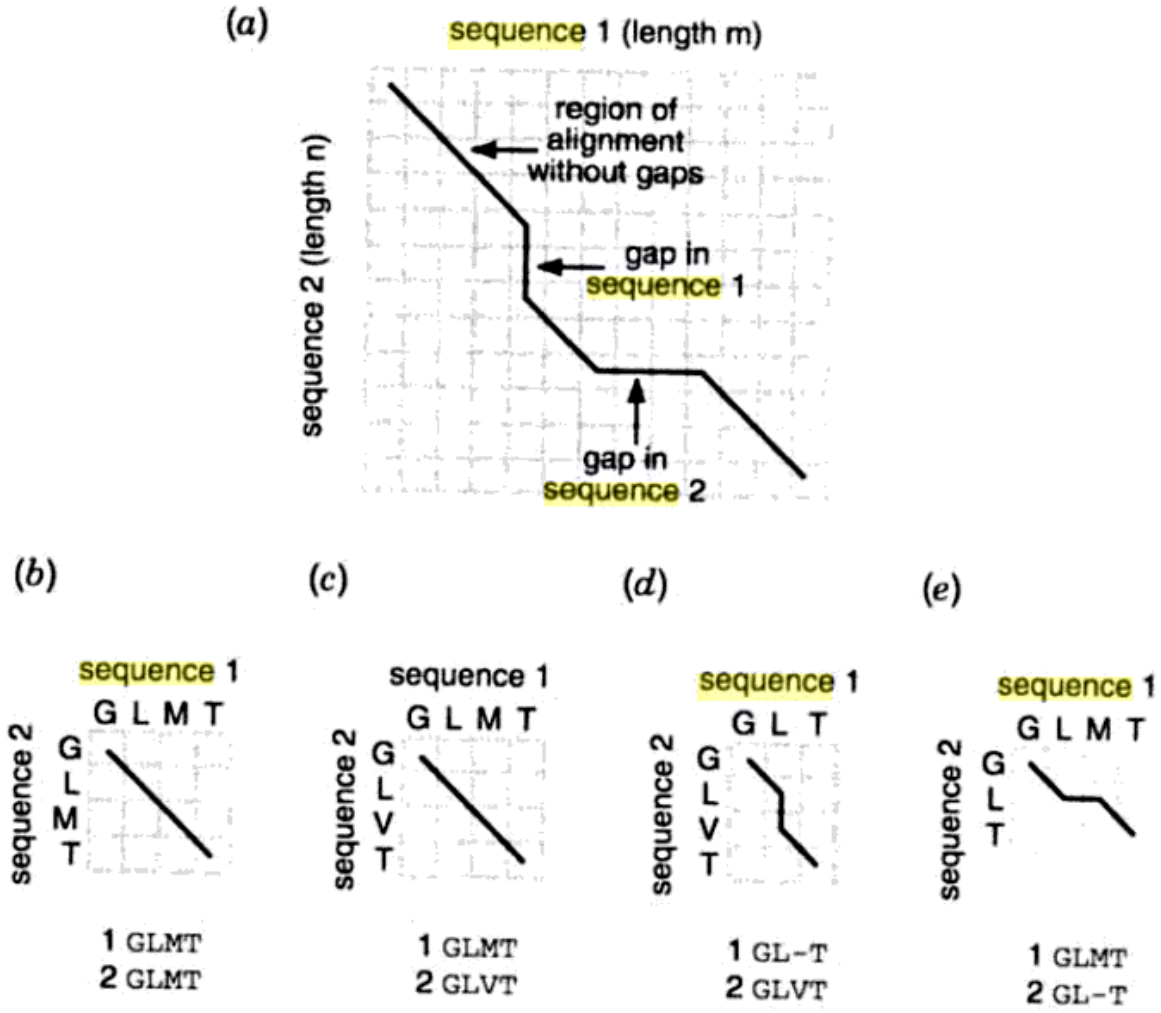
- from traceback t to alignment (a', b')

  a': by substituting a_i for the ith occurrence of $\nwarrow$ or $\uparrow$, and by replacing all occurrences of $\leftarrow$ with $-$.

  b': by substituting b_j for the jth occurrence of $\leftarrow$ or $\nwarrow$, and by replacing all occurrences of $\uparrow$ with $-$.
- $a = AT$ and $b = AAGT$

$$\begin{array}{cccc} \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ A & - & - & T \\ A & A & G & T \end{array} \qquad \begin{array}{cccc} \leftarrow & \nwarrow & \leftarrow & \nwarrow \\ - & A & - & T \\ A & A & G & T \end{array}$$

Understanding the traceback matrix



[Pevsner, Bioinformatics and functional genomics, Wiley 2003]

Final example

x=TTCATA, y=TGCTCGTA
scoring: +5 for match, -2 for mismatch and -6 for each in/del.

Dynamic programming matrix:

j → (sequence **y**)

i		0	1 T	2 G	3 C	4 T	5 C	6 G	7 T	8 = N A
0		0	–6	–12	–18	–24	–30	–36	–42	–48
1	T	–6	5	–1	–7	–13	–19	–25	–31	–37
2	T	–12	–1	3	–3	–2	–8	–14	–20	–26
3	C	–18	–7	–3	8	2	3	–3	–9	–15
4	A	–24	–13	–9	2	6	0	1	–5	–4
5	T	–30	–19	–15	–4	7	4	–2	6	0
M = 6	A	–36	–25	–21	–10	1	5	2	0	11

i ↓ (sequence **x**)

Optimum alignment scores 11:

T	**–**	**–**	**T**	**C**	**A**	**T**	**A**
T	**G**	**C**	**T**	**C**	**G**	**T**	**A**
+5	–6	–6	+5	+5	–2	+5	+5

[Eddy, Nat. Biotech., 2004]

Conclusion & Outlook

- We have seen how to compute the pairwise edit distance and the corresponding optimal global alignment (Needleman/Wunsch algorithm).
- Essentials: metric distance measures, edit distance equals alignment distance, dynamic programming, additivity and alignment decomposition
- Before going multiple, we will look at variations on the theme
  - More realistic, non-linear gap costs and
  - Similarity scores and local alignment