UNIVERSITÄT
DES
SAARLANDES

ZBI ZENTRUM FÜR BIOINFORMATIK

Pairwise Sequence Alignments

Algorithms for Sequence Analysis

Sven Rahmann

Summer 2021

Overview

Previous Lectures

- Distance and similarity measures between two sequences
- Error-tolerant pattern search (edit distance) in a text:
  Algorithms: Basic DP, Ukkonen, Myers, NFA-Shift-And, NFA-FM, Four Russians.
- Alignments as visualization of edit process (global, semiglobal)

Overview

Previous Lectures

- Distance and similarity measures between two sequences
- Error-tolerant pattern search (edit distance) in a text:
  Algorithms: Basic DP, Ukkonen, Myers, NFA-Shift-And, NFA-FM, Four Russians.
- Alignments as visualization of edit process (global, semiglobal)

Today

- From costs (distances) to scores (similarities)
- General scoring schemes
- More general introduction of alignments
- Four variants of alignments:
  (1) global, (2) semiglobal (pattern search),
  (3) free end gaps (overlap detection), (4) local (regions of similarity)

Scoring Schemes for Pairwise Sequence Comparison

Need for fine-grained similarity

- Comparison of biosequences (esp. protein sequences) needs a fine-grained notion of similarity instead of only “equal” vs. “not equal” amino acids.
- **Example:** Leucine (L) and Isoleucine (I) are physically and chemically similar. Tryptophan (W) has very different properties than most other amino acids.

Scoring Schemes for Pairwise Sequence Comparison

Need for fine-grained similarity

- Comparison of biosequences (esp. protein sequences) needs a fine-grained notion of similarity instead of only “equal” vs. “not equal” amino acids.
- **Example:** Leucine (L) and Isoleucine (I) are physically and chemically similar. Tryptophan (W) has very different properties than most other amino acids.

Change of paradigm: Zero-centered similarity

- Evaluate similarity (positive and negative) instead of distances (non-negative)
- Value of 0 means “neutral”, positive means “similar”, negative means ”dissimilar”.

Scoring Schemes for Pairwise Sequence Comparison

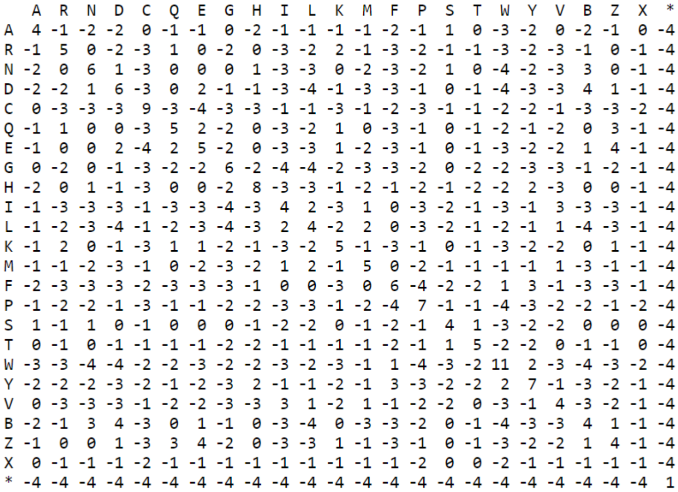
Need for fine-grained similarity

- Comparison of biosequences (esp. protein sequences) needs a fine-grained notion of similarity instead of only “equal” vs. “not equal” amino acids.
- **Example:** Leucine (L) and Isoleucine (I) are physically and chemically similar. Tryptophan (W) has very different properties than most other amino acids.

Change of paradigm: Zero-centered similarity

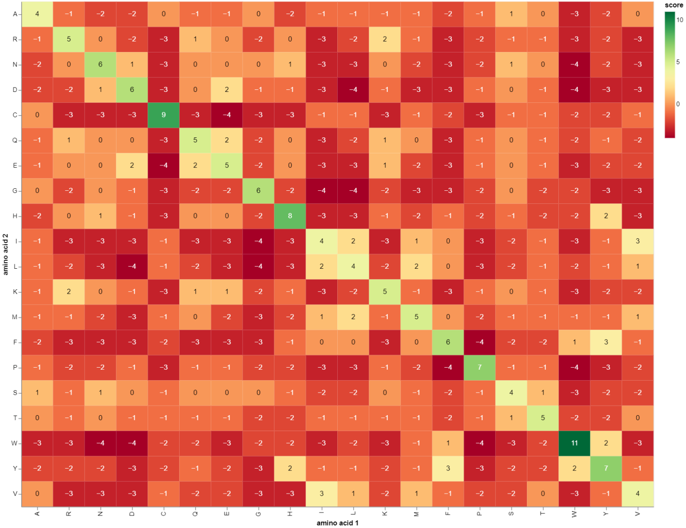
- Evaluate similarity (positive and negative) instead of distances (non-negative)
- Value of 0 means “neutral”, positive means “similar”, negative means ”dissimilar”.
- **Therefore:** Use a general **score matrix** $s = s(a, b)$ for any $a, b \in \Sigma$, and (negative) similarity values (**gap scores**) for insertions and deletions.

Example: BLOSUM62 Scoring Matrix for Amino Acids



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1	-4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1	-4
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-4
*	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1

Example: BLOSUM62 Scoring Matrix for Amino Acids



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	−1	−2	−2	0	−1	−1	0	−2	−1	−1	−1	−1	−2	−1	1	0	−3	−2	0
R	−1	5	0	−2	−3	1	0	−2	0	−3	−2	2	−1	−3	−2	−1	−1	−3	−2	−3
N	−2	0	6	1	−3	0	0	0	1	−3	−3	0	−2	−3	−2	1	0	−4	−2	−3
D	−2	−2	1	6	−3	0	2	−1	−1	−3	−4	−1	−3	−3	−1	0	−1	−4	−3	−3
C	0	−3	−3	−3	9	−3	−4	−3	−3	−1	−1	−3	−1	−2	−3	−1	−1	−2	−2	−1
Q	−1	1	0	0	−3	5	2	−2	0	−3	−2	1	0	−3	−1	0	−1	−2	−1	−2
E	−1	0	0	2	−4	2	5	−2	0	−3	−3	1	−2	−3	−1	0	−1	−3	−2	−2
G	0	−2	0	−1	−3	−2	−2	6	−2	−4	−4	−2	−3	−3	−2	0	−2	−2	−3	−3
H	−2	0	1	−1	−3	0	0	−2	8	−3	−3	−1	−2	−1	−2	−1	−2	−2	2	−3
I	−1	−3	−3	−3	−1	−3	−3	−4	−3	4	2	−3	1	0	−3	−2	−1	−3	−1	3
L	−1	−2	−3	−4	−1	−2	−3	−4	−3	2	4	−2	2	0	−3	−2	−1	−2	−1	1
K	−1	2	0	−1	−3	1	1	−2	−1	−3	−2	5	−1	−3	−1	0	−1	−3	−2	−2
M	−1	−1	−2	−3	−1	0	−2	−3	−2	1	2	−1	5	0	−2	−1	−1	−1	−1	1
F	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	6	−4	−2	−2	1	3	−1
P	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	7	−1	−1	−4	−3	−2
S	1	−1	1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	4	1	−3	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	1	5	−2	−2	0
W	−3	−3	−4	−4	−2	−2	−3	−2	−2	−3	−2	−3	−1	1	−4	−3	−2	11	2	−3
Y	−2	−2	−2	−3	−2	−1	−2	−3	2	−1	−1	−2	−1	3	−3	−2	−2	2	7	−1
V	0	−3	−3	−3	−1	−2	−2	−3	−3	3	1	−2	1	−1	−2	−2	0	−3	−1	4

Reminder: Alignments

Definition (Alignment, Projections π_1, π_2)

An **alignment** is a string A over the **alignment alphabet** $(\Sigma \cup \{-\})^2 \setminus \{(-,-)\}$ (pairs of characters, or one character paired with a gap).
The **first (second) projection** π_1 (π_2) reads the first (second) elements without gaps, so π_1 is the string homomorphism with $\pi_1\big((a,b)\big) := a$ and $\pi_1\big((-,b)\big) := \epsilon$, etc.

Reminder: Alignments

Definition (Alignment, Projections π_1, π_2)

An **alignment** is a string A over the **alignment alphabet** $(\Sigma \cup \{-\})^2 \setminus \{(-,-)\}$ (pairs of characters, or one character paired with a gap).
The **first (second) projection** π_1 (π_2) reads the first (second) elements without gaps, so π_1 is the string homomorphism with $\pi_1\big((a,b)\big) := a$ and $\pi_1\big((-,b)\big) := \epsilon$, etc.

Definition (Global alignment)

A **global alignment** between $s, t \in \Sigma^*$ is an alignment with $\pi_1(A) = s$, $\pi_2(A) = t$.

Reminder: Alignments

Definition (Alignment, Projections π_1, π_2)

An **alignment** is a string A over the **alignment alphabet** $(\Sigma \cup \{-\})^2 \setminus \{(-,-)\}$ (pairs of characters, or one character paired with a gap).
The **first (second) projection** π_1 (π_2) reads the first (second) elements without gaps, so π_1 is the string homomorphism with $\pi_1\big((a,b)\big) := a$ and $\pi_1\big((-,b)\big) := \epsilon$, etc.

Definition (Global alignment)

A **global alignment** between $s, t \in \Sigma^*$ is an alignment with $\pi_1(A) = s$, $\pi_2(A) = t$.

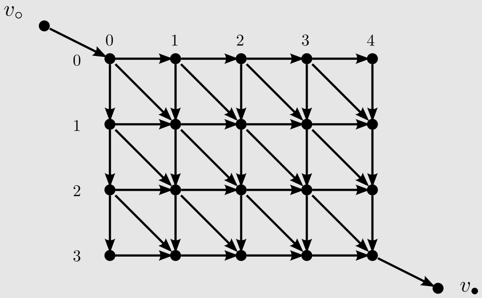
Definition (Semiglobal alignment)

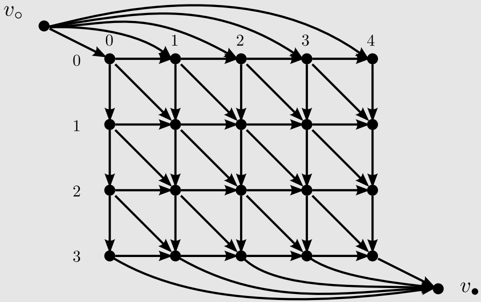
A **semiglobal alignment** between $P, T \in \Sigma^*$ is an alignment
with $\pi_1(A) = P$, $\pi_2(A) = T'$, where T' is any substring of T.

Universal Alignment Algorithm

Given

- Sequences s, t
- Scoring scheme
- Alignment graph topology (e.g., for global or semiglobal alignment)





Universal Alignment Algorithm

- **Given**: sequences s, t, scoring scheme, graph topology
- **Sought:**
  - Maximum score among all paths $v_\circ \to v_\bullet$ (**optimal alignment score**)
  - A path that maximizes the scores (**optimal alignment**)
- Let $S(v)$ be the maximal score of all paths $v_\circ \to v$, and $S(v_\circ) := 0$.
- Let $T(v)$ be the predecessor of v, from which the maximum $S(v)$ is obtained.

Universal Alignment Algorithm

- **Given**: sequences s, t, scoring scheme, graph topology
- **Sought:**
  - Maximum score among all paths $v_\circ \to v_\bullet$ (**optimal alignment score**)
  - A path that maximizes the scores (**optimal alignment**)
- Let $S(v)$ be the maximal score of all paths $v_\circ \to v$, and $S(v_\circ) := 0$.
- Let $T(v)$ be the predecessor of v, from which the maximum $S(v)$ is obtained.
- For $v \neq v_\circ$:

$$S(v) = \max_{w\colon w \to v \in E} \{S(w) + \mathrm{score}(w \to v)\},$$

$$T(v) = \operatorname*{arg\,max}_{w\colon w \to v \in E} \{S(w) + \mathrm{score}(w \to v)\}.$$

- Compute nodes in topological order (graph is acyclic!)
- The optimal score is obtained as $S(v_\bullet)$.
- The optimal path (alignment) is obtained by traceback from $v_\bullet$: $v_\bullet \to T(v_\bullet) \to T(T(v_\bullet)) \to \cdots \to T^k(v_\bullet) \to \cdots \to v_\circ$.

Traceback

Traceback, also **Backtracing**

- Reconstruction of the optimal path by tracing back the predecessor nodes that lead to the optimal score value in each node
- Do not confuse with Backtracking!

Traceback

Traceback, also **Backtracing**

- Reconstruction of the optimal path by tracing back the predecessor nodes that lead to the optimal score value in each node
- Do not confuse with Backtracking!

Optimal path and alignment

- Optimal path is reconstructed backwards, can be flipped when done
- Optimal alignment:
  Read off the edge labels along the optimal path

Traceback

Traceback, also **Backtracing**

- Reconstruction of the optimal path by tracing back the predecessor nodes that lead to the optimal score value in each node
- Do not confuse with Backtracking!

Optimal path and alignment

- Optimal path is reconstructed backwards, can be flipped when done
- Optimal alignment:
  Read off the edge labels along the optimal path

Time and memory requirements

- **Running time:** $O(m+n)$ for an $m \times n$ matrix (maximum length of a path)
- **Memory:** $O(mn)$ because the full matrix T must be stored (improvement soon)

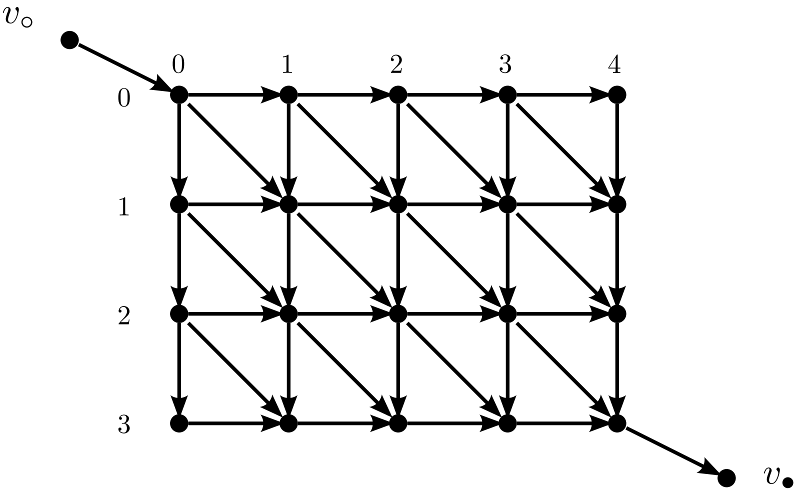
Variants of Alignments

Four Variants

1. Global alignment (similarity of full sequences)
2. Semiglobal alignment (pattern search)
3. Free end gaps alignment (good/optimal overlap)
4. local alignment (region[s] of high/optimal similarity)

In the following, we discuss the associated **graph topology** for each variant.
All variants can be handled uniformly with the **universal alignment algorithm**.

Global Alignment



Global Alignment

Definition (global alignment graph)

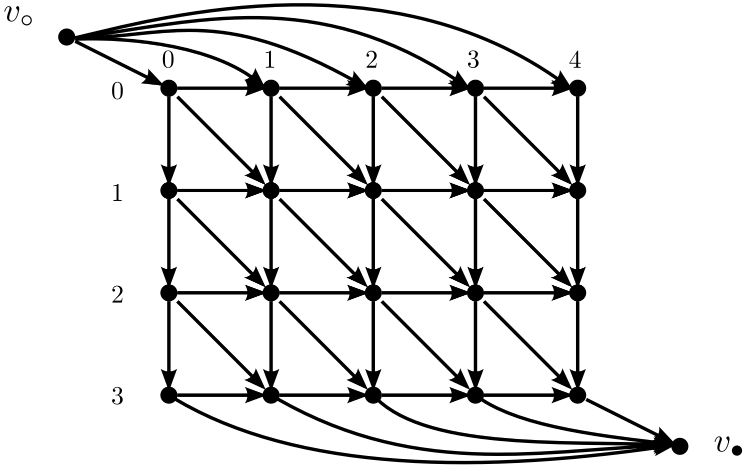
- Nodes $V := \{(i,j) : 0 \leq i \leq m, 0 \leq j \leq n\} \cup \{v_\circ, v_\bullet\}$
- Edges:

	Edge	label	**score**
horizontal	$(i,j) \to (i,j+1)$	$\begin{bmatrix}-\\t_j\end{bmatrix}$	< 0 (*)
vertical	$(i,j) \to (i+1,j)$	$\begin{bmatrix}s_i\\-\end{bmatrix}$	< 0 (*)
diagonal	$(i,j) \to (i+1,j+1)$	$\begin{bmatrix}s_i\\t_j\end{bmatrix}$	beliebig (*)
Initialization	$v_\circ \to (0,0)$	ϵ	0
Finalization	$(m,n) \to v_\bullet$	ϵ	0

(*): Meaningful scoring schemes have negative scores for gaps and most substitutions, and positive scores for identities.

Semiglobal Alignment (Pattern Search)

Additional initialization edges $v_\circ \to (0,j)$ and finalization edges $(m,j) \to v_\bullet$:



“Free End Gaps” Alignment (Overlap Detection)

Question

- (How) Do two sequences overlap?

- Gaps (overhangs) at either border of either sequence shall not be penalized.

"Free End Gaps" Alignment (Overlap Detection)

Question

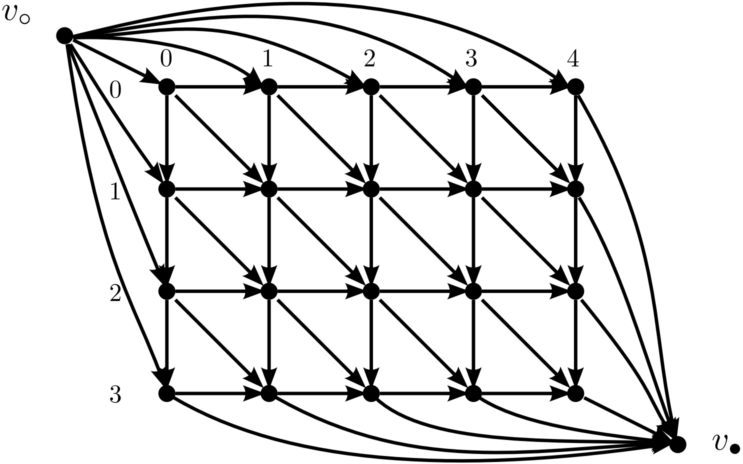
- (How) Do two sequences overlap?

- Gaps (overhangs) at either border of either sequence shall not be penalized.

Graph construction

Additional initialization edges $v_\circ \rightarrow (i, 0)$ and $v_\circ \rightarrow (0, j)$,
and finalization edges $(i, n) \rightarrow v_\bullet$ and $(m, j) \rightarrow v_\bullet$.
(All such edges have empty labels and contribute score 0.)

“Free End Gaps” Alignment (Overlap Detection)



Local Alignment

Question

- (Where) Are there regions (substrings) of high similarity between two sequences?
- Where are the most similar substrings (maximal score) ?

- Formally: Find alignment with maximal score among all substrings s' of s and t' of t.

Local Alignment

Question

- (Where) Are there regions (substrings) of high similarity between two sequences?
- Where are the most similar substrings (maximal score) ?

- Formally: Find alignment with maximal score among all substrings s' of s and t' of t.

Graph construction

- Initialization edges $v_\circ \to (i,j)$ for all $0 \leq i \leq m$, $0 \leq j \leq n$,
- Finalization edges $(i,j) \to v_\bullet$ for all $0 \leq i \leq m$, $0 \leq j \leq n$.
- Visualization is not helpful (far too many edges)

Variants and Distances vs. Scores

Meaningful combinations

Variant	**Distances**	**Scores**
Global alignment (similarity of full sequences)	✓	✓
Semiglobal alignment (pattern search)	✓	✓
Free end gap alignment (good/optimal overlap)		✓
Local alignment (region[s] of high/optimal similarity)		✓

Variants and Distances vs. Scores

Meaningful combinations

Variant	**Distances**	**Scores**
Global alignment (similarity of full sequences)	✓	✓
Semiglobal alignment (pattern search)	✓	✓
Free end gap alignment (good/optimal overlap)		✓
Local alignment (region[s] of high/optimal similarity)		✓

Why?

Optimal distance is always zero ($d \geq 0$).
Free end gap and local alignments allow trivial “empty” alignments,
which always have distance zero. No incentive for non-trivial alignments.

Specialization of Algorithms

For each alignment variant (graph topology):

- What is the interpretation of the score $S(v)$ for any $v = (i, j)$?
  (“Score of an optimal alignment of . . . ”)
- How does the universal algorithm specialize to matrix form ?
  First row and column ?
  $S[i, j] = \max\{\dots\}$?
  Collection of interesting results or optimal result ?
- How do running time and memory requirements change vs. global alignment?
  (They don’t.)

Specialization of Algorithms

For each alignment variant (graph topology):

- What is the interpretation of the score $S(v)$ for any $v = (i, j)$?
  (“Score of an optimal alignment of . . . ”)
- How does the universal algorithm specialize to matrix form ?
  First row and column ?
  $S[i, j] = \max\{\dots\}$?
  Collection of interesting results or optimal result ?
- How do running time and memory requirements change vs. global alignment?
  (They don’t.)

Algorithm Names

- Global alignment: Needleman-Wunsch algorithm (NW)
- Local alignment: Smith-Waterman alignment (SW)

Needleman-Wunsch Algorithm (Score-Based, Global, Full Matrix)

```python
def needleman_wunsch(s, t, score):
    m, n, gapscore = len(s), len(t), score(None)
    S = np.zeros((m+1, n+1), dtype=np.int32)  # scores
    T = np.zeros((m+1, n+1), dtype=np.uint8)  # traceback
    S[0,:] = np.arange(n+1, dtype=S.dtype) * gapscore
    S[:,0] = np.arange(m+1, dtype=S.dtype) * gapscore
    T[0,0] = HOME; T[0,1:] = HORIZONTAL; T[1:,0] = VERTICAL
    for i, si in zip(count(1), s):  # (row, character in s)
        for j, tj in zip(count(1), t):  # (col, character in t)
            d = S[i-1, j-1] + score(si, tj)
            h = S[i, j-1] + gapscore
            v = S[i-1, j] + gapscore
            S[i,j] = opt = max(d, h, v)
            T[i,j] = (d==opt)*DIAGONAL + (h==opt)*HORIZONTAL \
                     + (v==opt)*VERTICAL
    return S[m,n], traceback(m, n, T, s, t)
```

Smith-Waterman Algorithm (Score-Based, Local, Full Matrix)

```python
def smith_waterman(s, t, score):
    m, n, gapscore = len(s), len(t), score(None)
    S = np.zeros((m+1, n+1), dtype=np.int32)  # scores
    T = np.zeros((m+1, n+1), dtype=np.uint8)  # traceback
    T[0,:] = HOME;  T[:,0] = HOME  # alignments end at border
    best = (-1, -1, -1)  # best (S, i, j)
    for i, si in zip(count(1), s):  # (row, character in s)
        for j, tj in zip(count(1), t):  # (col, character in t)
            d = S[i-1, j-1] + score(si, tj)
            h = S[i, j-1] + gapscore
            v = S[i-1, j] + gapscore
            S[i,j] = opt = max(0, d, h, v)  # note additional 0
            T[i,j] = (d==opt)*DIAGONAL + (h==opt)*HORIZONTAL \
                   + (v==opt)*VERTICAL  # can be HOME otherwise
            if S[i,j] > best[0]:  best = (S[i,j], i, j)
    result, i, j = best
    return result, traceback(i, j, T, s, t)
```

Implementation of Traceback

```python
HOME, DIAGONAL, HORIZONTAL, VERTICAL = 0, 1, 2, 4
def traceback(i, j, T, s, t, *, GAP='-'):
    # We reconstruct the alignment by traceback (T) from i, j
    As, At = [], []  # rows of alignment: As (for s), At (for t)
    while T[i,j] != HOME:
        trace = T[i,j]
        if (trace & DIAGONAL):
            i -= 1;  As.append(s[i])
            j -= 1;  At.append(t[j])
        elif (trace & HORIZONTAL):
            As.append(GAP)
            j -= 1; At.append(t[j])
        elif (trace & VERTICAL):
            i -= 1; As.append(s[i])
            At.append(GAP)
    # create the final alignment (pair of strings)
    return ("".join(As[::-1]), "".join(At[::-1]))
```

Summary

- Motivation of scoring schemes vs. cost functions
- Definition of pairwise alignments in general
- Definition of four pairwise sequence alignment variants
- Alignment graphs and four topology variants
- Universal alignment algorithm on graphs
- Universal traceback
- Specialization: Needleman-Wunsch (global)
- Specialization: Smith-Waterman (local)
- Other specialzations: Homework

Possible Exam Questions

- Define alignment (in general).
- Define global / semiglobal / etc. alignment of two strings s, t.
- Explain four variants of alignments and their applications / use cases.
- What is the difference between score and cost function and why is it important?
- Why can't we use costs for free end gap and local alignment?
- How can sequence alignment be formulated as a graph problem?
- Show the alignment graph topology for each variant.
- Explain the universal alignment algorithm on the alignment graph.
- Give the DP formulation for computing an alignment score (any variant).
- Compute an optimal alignment (any variant) for two given strings.
- Explain traceback.