UNIVERSITÄT
DES
SAARLANDES

Extensions and Improvements of Pairwise Sequence Alignment

Algorithms for Sequence Analysis

Sven Rahmann

Summer 2021

Overview

Previously: Pairwise Sequence Alignment

- Four variants: global, semiglobal, overlapping, local
- Theory of score matrices
- Local alignment statistics

Overview

Previously: Pairwise Sequence Alignment

- Four variants: global, semiglobal, overlapping, local
- Theory of score matrices
- Local alignment statistics

Today's Lecture: Extensions and Improvements

- Generalizing gap cost functions (especially affine gap costs)
- Alignments with anchor points
- Linear-space alignment (with traceback): Hirschberg's technique
- Conceptual problems with local alignments
- Alternative: Length-normalized alignments

Affine Gap Costs

On the Distribution of Gaps

- Gaps are insertions or deletions.
- Until now, a gap of length ℓ incurred a (negative) score of $g(\ell) = -\gamma \cdot \ell$, where $\gamma \geq 0$ is the **gap penalty**.
- Linear gap costs are not realisitic for biological sequences.

```
attccgacagaaagatac        attccgacagaaagatac
att-c--c-g----atac   vs.  attccg--------atac
```

On the Distribution of Gaps

- Gaps are insertions or deletions.
- Until now, a gap of length ℓ incurred a (negative) score of $g(\ell) = -\gamma \cdot \ell$, where $\gamma \geq 0$ is the **gap penalty**.
- Linear gap costs are not realisitic for biological sequences.

```
attccgacagaaagatac        attccgacagaaagatac
att-c--c-g----atac   vs.  attccg--------atac
```

- Opening a gap should be more expensive than extending a gap: results in fewer, but longer gaps.

Gap Costs

Special and general gap cost functions

- **Linear gap costs**: $g(\ell) = -\gamma \cdot \ell$
- **Affine gap costs**: $g(\ell) = -c - \gamma(\ell - 1)$
- **Convex gap costs**: general but convex $g(\ell)$
- **General gap costs**: general function $g(\ell)$

Parameters for affine gap costs

- ℓ: gap (=indel) length
- γ: gap extension penalty
- c: gap open penalty; assume $c \geq \gamma > 0$

Gap Costs

Special and general gap cost functions

- **Linear gap costs**: $g(\ell) = -\gamma \cdot \ell$
- **Affine gap costs**: $g(\ell) = -c - \gamma(\ell - 1)$
- **Convex gap costs**: general but convex $g(\ell)$
- **General gap costs**: general function $g(\ell)$

Parameters for affine gap costs

- ℓ: gap (=indel) length
- γ: gap extension penalty
- c: gap open penalty; assume $c \geq \gamma > 0$

How does generalizing the gap cost function affect the running time ?

Global Alignment with Affine Gap Costs

- **Goal:** Algorithm for affine gap costs with running time $O(mn)$
  (same as for linear gap costs; more general models are more expensive!)

Global Alignment with Affine Gap Costs

- **Goal:** Algorithm for affine gap costs with running time $O(mn)$ (same as for linear gap costs; more general models are more expensive!)
- **Idea:** Different types of (conditional) scores, similar to (DFA) states

Global Alignment with Affine Gap Costs

- **Goal:** Algorithm for affine gap costs with running time $O(mn)$ (same as for linear gap costs; more general models are more expensive!)
- **Idea:** Different types of (conditional) scores, similar to (DFA) states
- $S[i,j] := \max\{score(A) \mid A \text{ is alignment of } s[:i] \text{ and } t[:j]\}$.
- $V[i,j] := \max\left\{score(A) \mid \begin{array}{l} A \text{ is alignment of } s[:i] \text{ and } t[:j] \\ \text{ending with a gap } (-) \text{ in } t \end{array}\right\}$,
- $H[i,j] := \max\left\{score(A) \mid \begin{array}{l} A \text{ is alignment of } s[:i] \text{ und } t[:j] \\ \text{ending with a gap } (-) \text{ in } s \end{array}\right\}$.

Global Alignment with Affine Gap Costs

- **Goal:** Algorithm for affine gap costs with running time $O(mn)$ (same as for linear gap costs; more general models are more expensive!)
- **Idea:** Different types of (conditional) scores, similar to (DFA) states
- $S[i,j] := \max\{score(A) \mid A \text{ is alignment of } s[:i] \text{ and } t[:j]\}$.
- $V[i,j] := \max\left\{score(A) \mid \begin{array}{l} A \text{ is alignment of } s[:i] \text{ and } t[:j] \\ \text{ending with a gap } (-) \text{ in } t \end{array}\right\}$,
- $H[i,j] := \max\left\{score(A) \mid \begin{array}{l} A \text{ is alignment of } s[:i] \text{ und } t[:j] \\ \text{ending with a gap } (-) \text{ in } s \end{array}\right\}$.

$$V[i,j] = \max\{S[i-1,j] - c,\ V[i-1,j] - \gamma\},$$
$$H[i,j] = \max\{S[i,j-1] - c,\ H[i,j-1] - \gamma\},$$
$$S[i,j] = \max\{S[i-1,j-1] + \text{score}(s[i-1], t[j-1]),\ V[i,j],\ H[i,j]\}.$$

Global Alignment with Affine Gap Costs

Initialization

$$
\begin{aligned}
S[0,0] &= 0, \\
S[i,0] = V[i,0] &= g(i) \text{ for all } 1 \leq i \leq m, \\
S[0,j] = H[0,j] &= g(j) \text{ for all } 1 \leq j \leq n, \\
H[i,0] &= -\infty \text{ for all } 0 \leq i \leq m, \\
V[0,j] &= -\infty \text{ for all } 0 \leq j \leq n.
\end{aligned}
$$

Global Alignment with Affine Gap Costs

Initialization

$$
\begin{aligned}
S[0,0] &= 0, \\
S[i,0] = V[i,0] &= g(i) \text{ for all } 1 \leq i \leq m, \\
S[0,j] = H[0,j] &= g(j) \text{ for all } 1 \leq j \leq n, \\
H[i,0] &= -\infty \text{ for all } 0 \leq i \leq m, \\
V[0,j] &= -\infty \text{ for all } 0 \leq j \leq n.
\end{aligned}
$$

Traceback

In principle unchanged, may want to consider more cases:

$$
\begin{aligned}
S[i,j] = \max \{ & S[i-1,j-1] + \text{score}(s[i-1], t[j-1]), \; V[i,j], \; H[i,j] \} \\
& \{ \quad \nwarrow \quad , \quad \uparrow \quad , \quad \leftarrow \quad \}
\end{aligned}
$$

Global Alignment with Affine Gap Costs

Initialization

$$
\begin{aligned}
S[0,0] &= 0, \\
S[i,0] = V[i,0] &= g(i) \text{ for all } 1 \leq i \leq m, \\
S[0,j] = H[0,j] &= g(j) \text{ for all } 1 \leq j \leq n, \\
H[i,0] &= -\infty \text{ for all } 0 \leq i \leq m, \\
V[0,j] &= -\infty \text{ for all } 0 \leq j \leq n.
\end{aligned}
$$

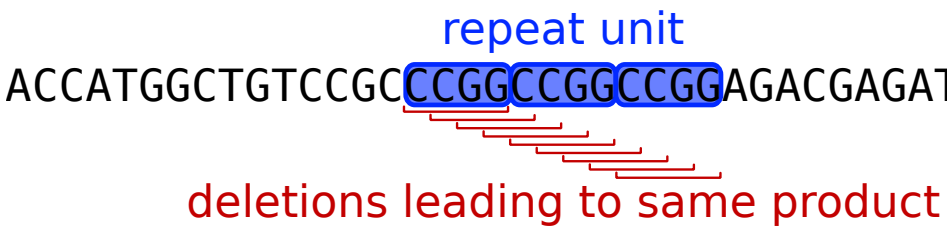
Traceback

In principle unchanged, may want to consider more cases:

$$
\begin{array}{c}
S[i,j] = \max\big\{S[i-1,j-1] + \text{score}(s[i-1], t[j-1]),\ V[i,j],\ H[i,j]\big\} \\
\{\qquad\qquad \nwarrow \qquad\qquad , \quad \uparrow \quad , \quad \leftarrow \quad \}
\end{array}
$$

Cost: Constant factor ($\times 3$) for time and memory

Side Note: Gap Shifting



Lemma: Gap Shifting

Let $s \in \Sigma^*$ contain a substring $r \in \Sigma^*$ that is repeated k times in tandem,
i.e., r^k is a substring of s with $r^k = s[i \dots i + |r|k - 1]$ for some i.
Then, deleting any length-$|r|$ substring from s within the interval yields the same result.

(Global) Alignment with Anchor Points

Global Alignments with Anchor Points

Question

- Find the best alignment whose path contains node (i, j) in the alignment graph.
- If (i, j) is not on a globally optimal path anyway,
  then such an alignment is suboptimal overall.

Global Alignments with Anchor Points

Question

- Find the best alignment whose path contains node (i, j) in the alignment graph.
- If (i, j) is not on a globally optimal path anyway, then such an alignment is suboptimal overall.

Solution

Compute two partial alignments, add their scores:

$$(0, 0) \to (i, j) \text{ and } (i, j) \to (m, n).$$

Global Alignments with Anchor Points

Question

- Find the best alignment whose path contains node (i, j) in the alignment graph.
- If (i, j) is not on a globally optimal path anyway, then such an alignment is suboptimal overall.

Solution

Compute two partial alignments, add their scores:

$$(0, 0) \to (i, j) \text{ and } (i, j) \to (m, n).$$

Scaling?

- Solve this problem **for all** (i, j) simultaneously.
- Iterating over the anchor point (i, j) yields total running time of $O(m^2\, n^2)$.
- Can we do it faster?

Global Alignments with Anchor Points

Improvement

- Optimal scores $(0,0) \to (i,j)$ for all (i,j) already exist: $S[i,j]$
- Scores $(i,j) \to (m,n)$ are scores $(0,0) \to (m-i, n-j)$ of the reverse strings.

Global Alignments with Anchor Points

Improvement

- Optimal scores $(0,0) \to (i,j)$ for all (i,j) already exist: $S[i,j]$
- Scores $(i,j) \to (m,n)$ are scores $(0,0) \to (m-i, n-j)$ of the reverse strings.
- Consider matrix $R[i,j]$ with optimal scores $(m,n) \to (i,j)$ (backwards)
- Careful with indexing!
- Then, sum $(S+R)[i,j]$ holds optimal score of all paths through (i,j)

Example

$s =$ andi and $t =$ handy.

	ϵ	h	a	n	d	y
ϵ	0	-1	-2	-3	-4	-5
a	-1	-1	0	-1	-2	-3
n	-2	-2	-1	1	0	-1
d	-3	-3	-2	0	2	1
i	-4	-4	-3	-1	1	1

1	2	0	-2	-4	-4	a
-1	0	1	-1	-3	-3	n
-3	-2	-1	0	-2	-2	d
-5	-4	-3	-2	-1	-1	i
-5	-4	-3	-2	-1	0	ϵ
h	a	n	d	y	ϵ	

Example

$s = \texttt{andi}$ and $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	0	-1	-2	-3	-4	-5
a	-1	-1	0	-1	-2	-3
n	-2	-2	-1	1	0	-1
d	-3	-3	-2	0	2	1
i	-4	-4	-3	-1	1	1

1	2	0	-2	-4	-4	a
-1	0	1	-1	-3	-3	n
-3	-2	-1	0	-2	-2	d
-5	-4	-3	-2	-1	-1	i
-5	-4	-3	-2	-1	0	ϵ
h	a	n	d	y	ϵ	

$\searrow \quad + \quad \swarrow$

1	1	-2	-5	-8	-9
-2	-1	1	-2	-5	-6
-5	-4	-2	1	-2	-3
-8	-7	-5	-2	1	0
-9	-8	-6	-3	0	1

Linear Space Alignment

Motivation: Linear Space Alignment

Pairwise alignment of two sequences $s \in \Sigma^m$ and $t \in \Sigma^n$ takes $O(nm)$ time and space.

Example

- Imagine $n = m = 3 \cdot 10^6$ (small bacterial genomes)
- The DP matrix would need $4nm = 36 \cdot 10^{12} = 36\text{T}$ bytes
- Assuming you can compute 1G table entries per second, you need around 2.5h.
- $\Rightarrow$ Time requirement much more manageable than memory requirement

Question

Can we be more memory efficient while being (almost) as fast?

Global Alignment in Linear Space

- We can compute the **optimal alignment score** in $O(\min(m, n))$ space.
- However, for the **optimal alignment** (traceback), we so far need $O(mn)$ space.

Global Alignment in Linear Space

- We can compute the **optimal alignment score** in $O(\min(m, n))$ space.
- However, for the **optimal alignment** (traceback), we so far need $O(mn)$ space.
- Daniel S. Hirschberg (1975) proposed an idea that avoids storing entire matrices, and reduces the space requirement to $O(m + n)$.
- The running time stays $O(mn)$; in practice, it doubles.

Global Alignment in Linear Space

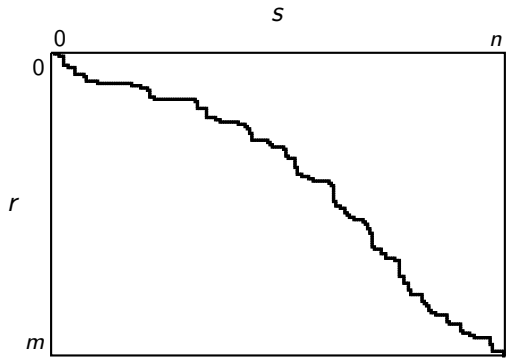
- We can compute the **optimal alignment score** in $O(\min(m, n))$ space.
- However, for the **optimal alignment** (traceback), we so far need $O(mn)$ space.
- Daniel S. Hirschberg (1975) proposed an idea that avoids storing entire matrices, and reduces the space requirement to $O(m + n)$.
- The running time stays $O(mn)$; in practice, it doubles.

Main Idea

Find out through which node $(i^*, n/2)$ the optimal path runs.
For known i^*, the problem then reduces to two smaller alignment problems:
upper left, lower right ("Divide-and-Conquer strategy").

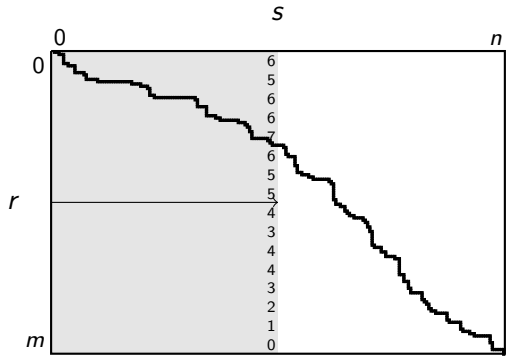
Hirschberg's Algorithm

Division into two subproblems:



Hirschberg's Algorithm

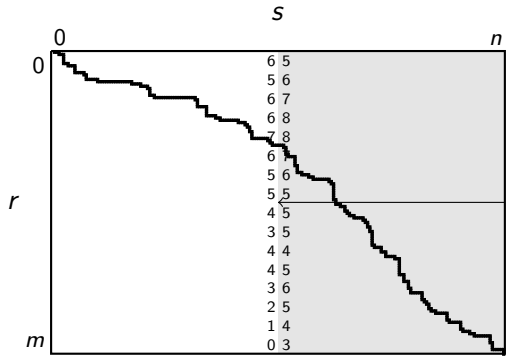
Division into two subproblems:



Align forward columns $0..n/2$

Hirschberg's Algorithm

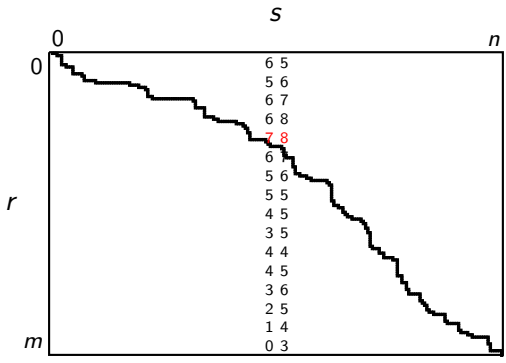
Division into two subproblems:



Align backwards columns $n..n/2$

Hirschberg's Algorithm

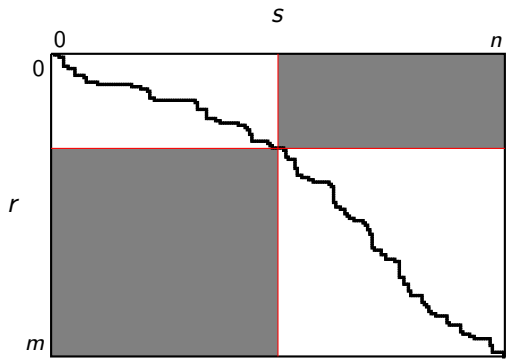
Division into two subproblems:



Find node $(i^*, n/2)$ with highest score in column $n/2$:
node is part of optimal alignment

Hirschberg's Algorithm

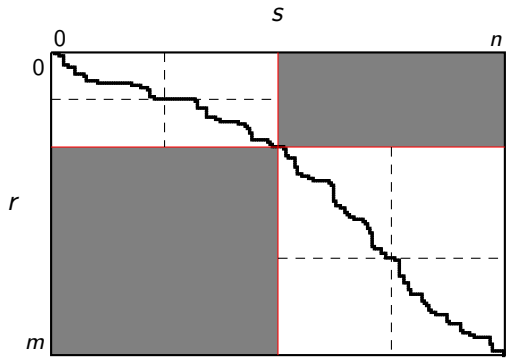
Division into two subproblems:



Recurse: Find sub-alignments $(0,0) \to (i^*, n/2)$ and $(i^*, n/2) \to (m,n)$

Hirschberg's Algorithm

Division into two subproblems:



Divide and recurse further until one string has length 1

Analysis of Hirschberg's Algorithm

Space

- Optimal path is determined from the center outward. Space: $O(m+n)$
- Space for score values: $O(\min(m,n))$ or $O(m+n)$
- Total space: $O(m+n)$

Analysis of Hirschberg's Algorithm

Space

- Optimal path is determined from the center outward. Space: $O(m+n)$
- Space for score values: $O(\min(m,n))$ or $O(m+n)$
- Total space: $O(m+n)$

Time

- In the first iteration, alignments $(0,0) \rightarrow (m, n/2)$ and $(0, n/2) \leftarrow (m,n)$ are computed: $O(mn)$ time

Analysis of Hirschberg's Algorithm

Space

- Optimal path is determined from the center outward. Space: $O(m+n)$
- Space for score values: $O(\min(m,n))$ or $O(m+n)$
- Total space: $O(m+n)$

Time

- In the first iteration, alignments $(0,0) \rightarrow (m, n/2)$ and $(0, n/2) \leftarrow (m,n)$ are computed: $O(mn)$ time
- After each iteration, half of the remaining matrix is removed.

Analysis of Hirschberg's Algorithm

Space

- Optimal path is determined from the center outward. Space: $O(m+n)$
- Space for score values: $O(\min(m,n))$ or $O(m+n)$
- Total space: $O(m+n)$

Time

- In the first iteration, alignments $(0,0) \rightarrow (m, n/2)$ and $(0, n/2) \leftarrow (m,n)$ are computed: $O(mn)$ time
- After each iteration, half of the remaining matrix is removed.
- Total running time: $\mathcal{O}(mn) \cdot (1 + 1/2 + 1/4 + \dots) \leq 2 \cdot \mathcal{O}(mn) = \mathcal{O}(mn)$

Summary: Hirschberg's Algorithm

- With Hirschberg's technique (divide-and-conquer, inside-out computation), we can determine an optimal global alignment in time $O(m + n)$.
- Asymptotic running time remains $O(mn)$, and doubles in practice.

Summary: Hirschberg's Algorithm

- With Hirschberg's technique (divide-and-conquer, inside-out computation), we can determine an optimal global alignment in time $O(m + n)$.
- Asymptotic running time remains $O(mn)$, and doubles in practice.
- Different alignment variants than global may use this technique, if we first determine the endpoints of the optimal alignment;
  Then: global alignment within its "box".
  Same time and space.

Problems with Local Alignment and Alternatives

Conceptual Problems with Local Alignment

- Score-based local alignment is widely used to discover regions of similarity between sequences (often proteins).
- The additive score function (alignment score = sum of scores over alignment columns) allows for efficient computation via DP:

$$score(A) := \sum_{0 \leq i < |A|} score(A_i)$$

Conceptual Problems with Local Alignment

- Score-based local alignment is widely used to discover regions of similarity between sequences (often proteins).
- The additive score function (alignment score = sum of scores over alignment columns) allows for efficient computation via DP:

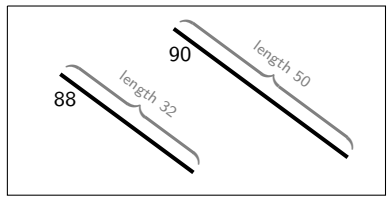
$$score(A) := \sum_{0 \leq i < |A|} score(A_i)$$

- However, additivity also leads to two problems:
  1. shadow effect
  2. mosaic effect

Conceptual Problems with Local Alignment

Definition (Shadow Effect)

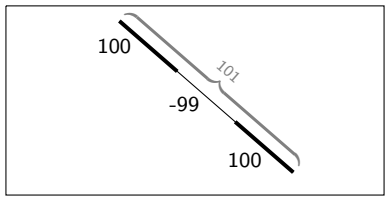
Longer alignments with many differences may get higher scores than ("shadow") shorter more exact alignments, even though the shorter alignment may be more interesting biologically.



Conceptual Problems with Local Alignment

Definition (Mosaic Effect)

In a long alignment with alternating regions of high/low/high similarity,
the whole alignment may receive a (slightly) higher score
than each of the shorter but well conserved regions individually.
The shorter more exact alignments would be more interesting biologically.



Length Normalized Alignment

Possible solution?

- Long alignments have an advantage over short ones; they can accumulate higher score.
- Attempt to define **length-normalized** score?

Length Normalized Alignment

Possible solution?

- Long alignments have an advantage over short ones; they can accumulate higher score.
- Attempt to define **length-normalized** score?
- Careful: $score(A)/|A|$ will favor length-1 alignments

Length Normalized Alignment

Possible solution?

- Long alignments have an advantage over short ones; they can accumulate higher score.
- Attempt to define **length-normalized** score?
- Careful: $score(A)/|A|$ will favor length-1 alignments

More pragmatic solution

Use a regularization parameter $L > 0$ corresponding to a minimum length

$$NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \le i < |A|} Score(A_i).$$

Length Normalized Alignment

Possible solution?

- Long alignments have an advantage over short ones; they can accumulate higher score.
- Attempt to define **length-normalized** score?
- Careful: $score(A)/|A|$ will favor length-1 alignments

More pragmatic solution

Use a regularization parameter $L > 0$ corresponding to a minimum length

$$NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \le i < |A|} Score(A_i).$$

Challenges: L needs to be known; DP algorithms don't apply (lack of additivity).

Length Normalized Alignment

Idea

$$\text{To maximize} \quad NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \le i < |A|} Score(A_i)\,,$$

use surrogate score function, for some fixed $\lambda > 0$:

$$\begin{aligned} DScore_{\lambda,L}(A) &:= Score(A) - \lambda(|A| + L) \\ &= -\lambda L + \sum_{i=1}^{|A|} (Score(A_i) - \lambda) \end{aligned}$$

Length Normalized Alignment

Idea

$$\text{To maximize} \quad NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \le i < |A|} Score(A_i)\,,$$

use surrogate score function, for some fixed $\lambda > 0$:

$$\begin{aligned} DScore_{\lambda,L}(A) &:= Score(A) - \lambda(|A| + L) \\ &= -\lambda L + \sum_{i=1}^{|A|} (Score(A_i) - \lambda) \end{aligned}$$

- allows to use known DP algorithm; all scores lowered by λ
- offset $-\lambda L$ irrelevant

Length Normalized Alignment

$$NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \le i < |A|} Score(A_i) \tag{1}$$

$$DScore_{\lambda,L}(A) := Score(A) - \lambda(|A| + L) \tag{2}$$

Relation between the two scores

- One can show that there exists $\lambda^* \geq 0$ such that the solutions coincide:
  - $\max_A DScore_{\lambda^*,L}(A)$
  - $\max_A NormScore_L(A)$

Length Normalized Alignment

$$NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \le i < |A|} Score(A_i) \tag{1}$$

$$DScore_{\lambda,L}(A) := Score(A) - \lambda(|A| + L) \tag{2}$$

Relation between the two scores

- One can show that there exists $\lambda^* \geq 0$ such that the solutions coincide:
  - $\max_A DScore_{\lambda^*,L}(A)$
  - $\max_A NormScore_L(A)$
- For given λ, we can efficiently solve (2) by DP, yielding alignment $A(\lambda)$
- We can evaluate $NormScore(A(\lambda))$ and search for the correct λ^*.
- In practice, binary search with 3–5 iteration suffices.

Length Normalized Alignment

$$NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \le i < |A|} Score(A_i) \tag{1}$$

$$DScore_{\lambda,L}(A) := Score(A) - \lambda(|A| + L) \tag{2}$$

Relation between the two scores

- One can show that there exists $\lambda^* \geq 0$ such that the solutions coincide:
  - $\max_A DScore_{\lambda^*,L}(A)$
  - $\max_A NormScore_L(A)$
- For given λ, we can efficiently solve (2) by DP, yielding alignment $A(\lambda)$
- We can evaluate $NormScore(A(\lambda))$ and search for the correct λ^*.
- In practice, binary search with 3–5 iteration suffices.
- $\Rightarrow$ $NormScore_L$ can be efficiently optimized (but rarely used in practice).

Summary

Extensions and Improvements

- Generalizing gap cost functions (especially affine gap costs)
- Alignments with affine gap costs in $O(mn)$ time
- Alignments with anchor points
- Linear-space alignment (with traceback): Hirschberg's technique
- Conceptual problems with local alignment
- Alternative: Length-normalized alignment

Possible Exam Questions

- What are linear vs. affine gap costs?
- Explain how to implement alignment with general gap costs (time?)
- Explain how to implement alignment with affine gap costs (time?)
- How can alignments (tracebacks) be obtained in $O(m+n)$ space instead of $O(mn)$ space? Illustrate why this is crucial in practice.
- How is the running time affected by linear-space traceback?
- Illustrate two conceptual problems with local alignment.
- Explain the idea of length-normalized local alignment.
- What is the role of the parameter $L > 0$?
- Why can't we use the standard DP algorithms for length-normalized alignment?
- How can we efficiently find the optimal length-normalized alignment?