# Bioinformatics I

Lecture 05:

**Alignment in Linear Space**
**and**
**Normalized Sequence Alignment**

`slides by` Dr. Sebastian Will

Bioinformatics Group, University Freiburg

Summer 2012

## Global Sequence Alignment in Linear Space

—

## The Algorithm of Hirschberg

## Recall Global Sequence Alignment

IN: sequences $a = a_1 \dots a_n$ and $b = b_1 \dots b_m$ in $\Sigma^*$

we defined: alignment $\mathcal{A} = (a^\diamond, b^\diamond)$ of $a$ and $b$

cost function $w(\mathcal{A}) = w(a^\diamond, b^\diamond)$

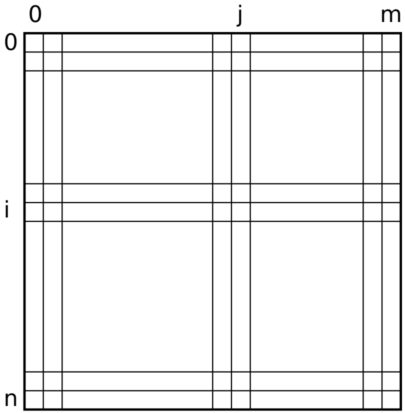
$$w(a^\diamond, b^\diamond) := \sum_{1 \leq k \leq |\mathcal{A}|} w(a^\diamond_k, b^\diamond_k)$$

$$w(x,y) = \begin{cases} \delta & x, y \in \Sigma, x \neq y \\ \gamma & x = - \text{ or } y = - \\ 0 & \text{otherwise} \end{cases}$$

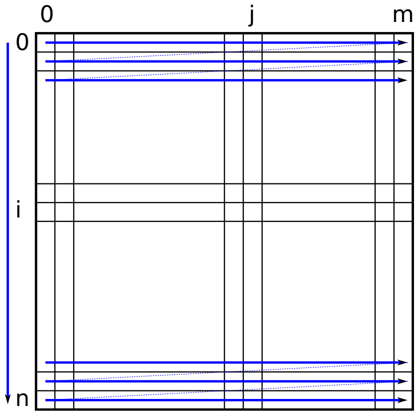
OUT: alignment $\mathcal{A}^*$ of $a$ and $b$ with minimal cost $w(\mathcal{A}^*)$

- Algorithm?
- Complexity?

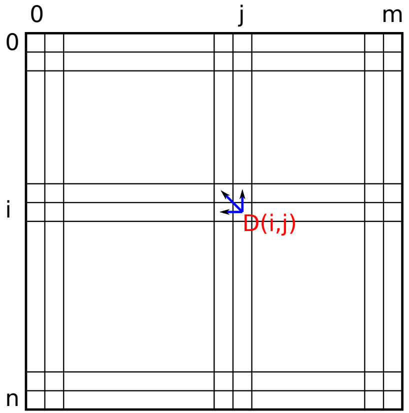
## Minimum Cost in $O(n^2)$ Space



# Minimum Cost in $O(n^2)$ Space

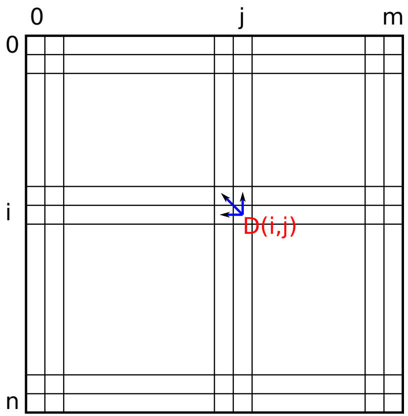


# Minimum Cost in $O(n^2)$ Space



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

## Minimum Cost in $O(n^2)$ Space



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

Observations:

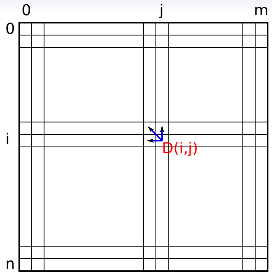
- Need only row $i-1$ to fill row $i$
- Can forget rows $i' < i-1$

$\Longrightarrow$ $D(n,m)$ in LINEAR SPACE

## Minimum Cost in Linear Space

*Pseudocode*

```
for $j := 0$ to $m$ do
    $D^0[j] := j\gamma$ // init row 0
end for
for $i := 1$ to $n$ do
    $D^1[0] := i\gamma$ // init column 0
    for $j := 1$ to $m$ do
        $D^1[j] := \min \begin{cases} D^0[j-1] + w(a_i, b_j) \\ D^0[j] + \gamma \\ D^1[j-1] + \gamma \end{cases}$
    end for
    swap vectors $D^0$ and $D^1$ //(!)
end for
```

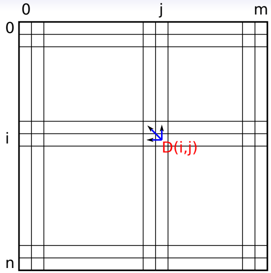


Stores only two vectors $D^0$ and $D^1$,
i.e. LINEAR SPACE

## Minimum Cost in Linear Space

*Pseudocode*

```
for $j := 0$ to $m$ do
    $D^0[j] := j\gamma$ // init row 0
end for
for $i := 1$ to $n$ do
    $D^1[0] := i\gamma$ // init column 0
    for $j := 1$ to $m$ do
        $D^1[j] := \min \begin{cases} D^0[j-1] + w(a_i, b_j) \\ D^0[j] + \gamma \\ D^1[j-1] + \gamma \end{cases}$
    end for
    swap vectors $D^0$ and $D^1$ //(!)
end for
```



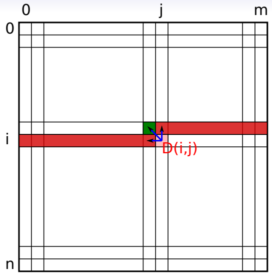
Stores only two vectors $D^0$ and $D^1$,
i.e. LINEAR SPACE

## Minimum Cost in Linear Space

*Pseudocode*

```
for $j := 0$ to $m$ do
    $D^0[j] := j\gamma$ // init row 0
end for
for $i := 1$ to $n$ do
    $D^1[0] := i\gamma$ // init column 0
    for $j := 1$ to $m$ do
        $D^1[j] := \min \begin{cases} D^0[j-1] + w(a_i, b_j) \\ D^0[j] + \gamma \\ D^1[j-1] + \gamma \end{cases}$
    end for
    swap vectors $D^0$ and $D^1$ //(!)
end for
```

Stores only two vectors $D^0$ and $D^1$, i.e. LINEAR SPACE

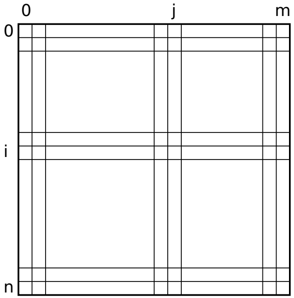


*Even less space:*
store only one row and one variable
(pseudocode not given)

# Best Alignment in Linear Space

- No direct traceback!
- Trace intersects row $i$: where?
  - Compute row $i$ in $O(n)$ space
    $\Rightarrow$ prefix alignment scores:
    $D(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_1,\dots,a_i$ and $b_1,\dots b_j\}$
  - Compute suffix alignment scores (1!)
    $D'(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_{i+1},\dots,a_n$ and $b_{j+1},\dots b_m\}$

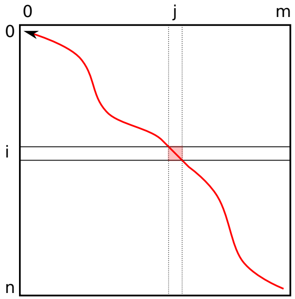
  *AND* combine with prefix alignment (2!)
    $\Rightarrow$ total alignment score "through $(i,j)$" is $D(i,j) + D'(i,j)$
  - look for minimum $D(i,j) + D'(i,j) \Rightarrow (i,j)$ on trace
- "naive" linear space alignment
  "do this for each row": $O(n^3)$ time!



# Best Alignment in Linear Space

- No direct traceback!
- Trace intersects row $i$: where?
    - Compute row $i$ in $O(n)$ space $\Rightarrow$ prefix alignment scores: $D(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_1,\dots,a_i$ and $b_1,\dots b_j\}$
    - Compute suffix alignment scores (1!) $D'(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_{i+1},\dots,a_n$ and $b_{j+1},\dots b_m\}$

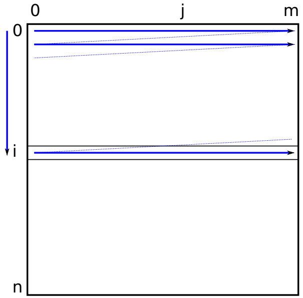
  *AND* combine with prefix alignment (2!) $\Rightarrow$ total alignment score "through $(i,j)$" is $D(i,j) + D'(i,j)$
    - look for minimum $D(i,j) + D'(i,j) \Rightarrow (i,j)$ on trace
- "naive" linear space alignment "do this for each row": $O(n^3)$ time!



## Best Alignment in Linear Space

- No direct traceback!
- Trace intersects row $i$: where?
  - Compute row $i$ in $O(n)$ space $\Rightarrow$ prefix alignment scores: $D(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A} \text{ alig. of } a_1,\dots,a_i \text{ and } b_1,\dots b_j\}$
  - Compute suffix alignment scores (1!) $D'(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A} \text{ alig. of } a_{i+1},\dots,a_n \text{ and } b_{j+1},\dots b_m\}$

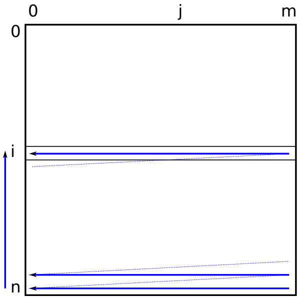
  *AND* combine with prefix alignment (2!) $\Rightarrow$ total alignment score "through $(i,j)$" is $D(i,j) + D'(i,j)$
  - look for minimum $D(i,j) + D'(i,j) \Rightarrow (i,j)$ on trace
- "naive" linear space alignment "do this for each row": $O(n^3)$ time!



# Best Alignment in Linear Space

- No direct traceback!
- Trace intersects row $i$: where?
  - Compute row $i$ in $O(n)$ space
    $\Rightarrow$ prefix alignment scores:
    $D(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_1, \dots, a_i$ and $b_1, \dots b_j\}$
  - Compute suffix alignment scores (1!)
    $D'(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_{i+1}, \dots, a_n$ and $b_{j+1}, \dots b_m\}$

  *AND* combine with prefix alignment (2!)
    $\Rightarrow$ total alignment score "through $(i,j)$" is $D(i,j) + D'(i,j)$
  - look for minimum $D(i,j) + D'(i,j) \Rightarrow$ $(i,j)$ on trace
- "naive" linear space alignment
  "do this for each row": $O(n^3)$ time!



(1!):

$D'(i,m) = i\gamma, \quad D'(n,j) = j\gamma$

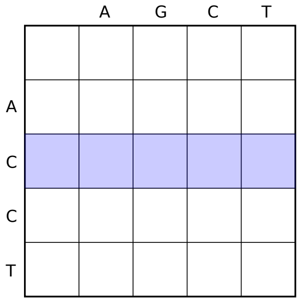
$$D'(i,j) = \min \begin{cases} D'(i+1,j+1) + w(a_{i+1}, b_{j+1}) \\ D'(i+1,j) + \gamma \\ D'(i,j+1) + \gamma \end{cases}$$

# Best Alignment in Linear Space

- No direct traceback!
- Trace intersects row $i$: where?
  - Compute row $i$ in $O(n)$ space
    $\Rightarrow$ prefix alignment scores:
    $D(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_1,\dots,a_i$ and $b_1,\dots b_j\}$
  - Compute suffix alignment scores (1!)
    $D'(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_{i+1},\dots,a_n$ and $b_{j+1},\dots b_m\}$

*AND* combine with prefix alignment (2!)
$\Rightarrow$ total alignment score "through $(i,j)$" is $D(i,j) + D'(i,j)$

  - look for minimum $D(i,j) + D'(i,j) \Rightarrow$ $(i,j)$ on trace
- "naive" linear space alignment
  "do this for each row": $O(n^3)$ time!



(2!): by Example
$a =$ACCT, $b =$AGCT, $i = 2$

$$D(2,3) = w\left(\begin{array}{l}\texttt{A-C}\\\texttt{AGC}\end{array}\right)$$

$$D'(2,3) = w\left(\begin{array}{l}\texttt{CT}\\\texttt{-T}\end{array}\right)$$

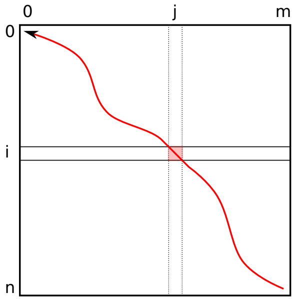
$$D(2,3) + D'(2,3) = w\left(\begin{array}{l}\texttt{A-CCT}\\\texttt{AGC-T}\end{array}\right)$$

# Best Alignment in Linear Space

- No direct traceback!
- Trace intersects row $i$: where?
    - Compute row $i$ in $O(n)$ space
      $\Rightarrow$ prefix alignment scores:
      $D(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_1, \dots, a_i$ and $b_1, \dots b_j\}$
    - Compute suffix alignment scores (1!)
      $D'(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_{i+1}, \dots, a_n$ and $b_{j+1}, \dots b_m\}$

  *AND* combine with prefix alignment (2!)
      $\Rightarrow$ total alignment score "through $(i,j)$" is $D(i,j) + D'(i,j)$
    - look for minimum $D(i,j) + D'(i,j) \Rightarrow (i,j)$ on trace

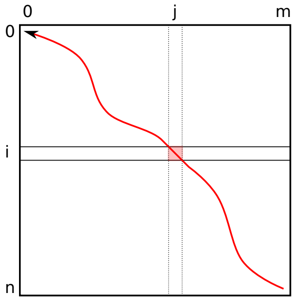
- "naive" linear space alignment
  "do this for each row": $O(n^3)$ time!



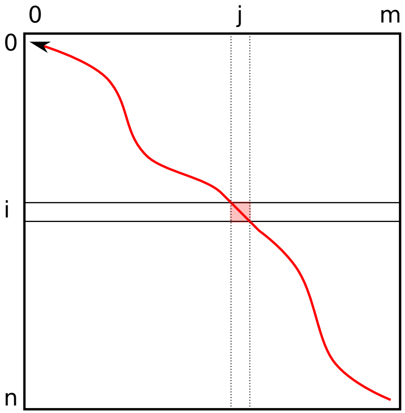
## Best Alignment in Linear Space

- No direct traceback!
- Trace intersects row $i$: where?
    - Compute row $i$ in $O(n)$ space
      $\Rightarrow$ prefix alignment scores:
      $D(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_1, \dots, a_i$ and $b_1, \dots b_j\}$
    - Compute suffix alignment scores (1!)
      $D'(i,j) = \min\{w(\mathcal{A}) \mid \mathcal{A}$ alig. of $a_{i+1}, \dots, a_n$ and $b_{j+1}, \dots b_m\}$

  *AND* combine with prefix alignment (2!)
      $\Rightarrow$ total alignment score "through $(i,j)$" is $D(i,j) + D'(i,j)$
    - look for minimum $D(i,j) + D'(i,j) \Rightarrow$ $(i,j)$ on trace
- "naive" linear space alignment
  "do this for each row": $O(n^3)$ time!



## Divide&Conquer: The Hirschberg Algorithm

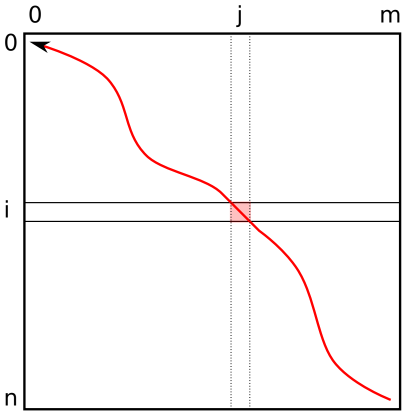


*pseudocode* TRACE($a, b$)

```
1. Find trace cell $(i, j)$ in row
   $i := \lceil n/2 \rceil$
2. Divide problem "find trace for
   $a_1 \dots a_n$ and $b_1 \dots b_m$" into
   subproblems
     a) TRACE($a_1 \dots a_{i-1}, b_1 \dots b_j$)
     b) TRACE($a_{i+1} \dots a_n, b_j \dots b_m$)
3. Solve subproblems recursively
   (terminate at $\leq 1$ rows)
```

Note: everything is done in LINEAR SPACE

## Divide&Conquer: The Hirschberg Algorithm

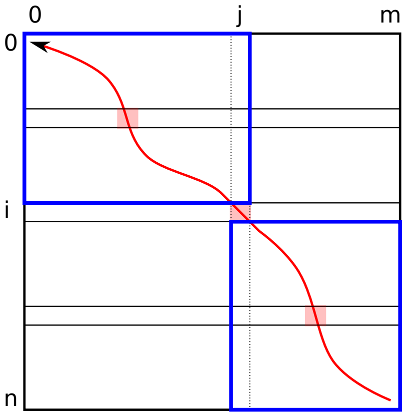


*pseudocode* TRACE($a$, $b$)

```
1. Find trace cell $(i, j)$ in row
   $i := \lceil n/2 \rceil$
2. Divide problem "find trace for
   $a_1 \dots a_n$ and $b_1 \dots b_m$" into
   subproblems
     a) TRACE($a_1 \dots a_{i-1}, b_1 \dots b_j$)
     b) TRACE($a_{i+1} \dots a_n, b_j \dots b_m$)
3. Solve subproblems recursively
   (terminate at $\leq 1$ rows)
```

Note: everything is done in LINEAR SPACE

# Divide&Conquer: The Hirschberg Algorithm

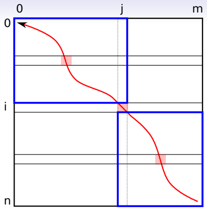


*pseudocode* TRACE($a$, $b$)

```
1. Find trace cell $(i, j)$ in row
   $i := \lceil n/2 \rceil$
2. Divide problem "find trace for
   $a_1 \dots a_n$ and $b_1 \dots b_m$" into
   subproblems
     a) TRACE($a_1 \dots a_{i-1}, b_1 \dots b_j$)
     b) TRACE($a_{i+1} \dots a_n, b_j \dots b_m$)
3. Solve subproblems recursively
   (terminate at $\leq 1$ rows)
```

Note: everything is done in LINEAR SPACE

## Hirschberg Runs in $O(n^2)$ Time



**Proof:**
by solving recursion for runtime $T(n,m)$:

$$T(0,m) = 0 \qquad T(1,m) = m$$

$$T(n,m) = nm + T(\lceil n/2 \rceil - 1, j) + T(n - \lceil n/2 \rceil - 1, m - j)$$

*Claim:* $T(n,m) \leq 2nm$

*Proof:* by induction on $n$
Ind. start: n=0,n=m □
Ind. step:

$$\begin{aligned} T(n,m) &\leq_{\text{(I.H.)}} nm + 2(\lceil n/2 \rceil - 1)j + 2(n - \lceil n/2 \rceil - 1)(m - j) \\ &\leq nm + 2(\lceil n/2 \rceil - 1)(j + m - j) \\ &< nm + nm \end{aligned}$$

□

## Conclusions: Linear space alignment

- Computing optimal alignment scores in linear space is simple
- One can compute prefix *and* suffix alignment scores by DP
- Hirschberg algorithm: linear space but only twice the time
- Divide and conquer: general algorithmic principle
- Run-time of D&C algorithm shown by induction
- Extensions possible, e.g. affine gap cost

## Normalized Local Sequence Alignment

[ Arslan et al., Bioinformatics 2001 ]

—

## Fractional Programming using Dinkelbach's Algorithm

## Recall Local Sequence Alignment

IN: sequences $a = a_1 \dots a_n$ and $b = b_1 \dots b_m$

define: a *local alignment* $\mathcal{A} = (a^\diamond, b^\diamond)$ *of a and b* is a (global) alignment of subsequences of $a$ and $b$

similarity function $s(\mathcal{A}) = s(a^\diamond, b^\diamond)$

$$s(a^\diamond, b^\diamond) := \sum_{1 \leq k \leq |\mathcal{A}|} s(a^\diamond_k, b^\diamond_k)$$

$$s(x,y) = \begin{cases} \gamma & x = - \text{ or } y = - \\ \sigma & x = y \\ \mu & x \neq y \end{cases}$$

OUT: local alignment $\mathcal{A}^*$ of $a$ and $b$ with maximum similarity $s(\mathcal{A}^*)$

## Local Alignment Example

- sequences
  $a =$GCATTUGCCUU
  $b =$CTTGACCATU
- similarity $s(x,y) = \begin{cases} -2 & x = - \text{ or } y = - \\ 3 & x = y \\ -1 & x \neq y \end{cases}$

- Local alignment with maximum similarity:
  (global alignment of subsequences $a_4, \dots, a_9$ and $b_2, \dots, b_7$)
  $\begin{array}{l} a^\diamond = \texttt{TTUG-CC} \\ b^\diamond = \texttt{TT-GACC} \end{array}$, $s(a^\diamond, b^\diamond) = 3 + 3 - 2 + 3 - 2 + 3 + 3 = 11$

- Another local alignment (g.a. of $a_2, a_3, a_4$ and $b_7, b_8, b_9$)
  $\mathcal{A} = \begin{pmatrix} \texttt{CAT} \\ \texttt{CAT} \end{pmatrix}$, $s(\mathcal{A}) = 3 + 3 + 3 = 9$

## Local Alignment Example

- sequences
  $a$ = GCA<u>TTUGCC</u>UU
  $b$ = C<u>TTGACC</u>ATU
- similarity $s(x,y) = \begin{cases} -2 & x = - \text{ or } y = - \\ 3 & x = y \\ -1 & x \neq y \end{cases}$
- Local alignment with maximum similarity:
  (global alignment of subsequences $a_4, \dots, a_9$ and $b_2, \dots, b_7$)

  $\begin{array}{l} a^\diamond = \texttt{TTUG-CC} \\ b^\diamond = \texttt{TT-GACC} \end{array}$, $s(a^\diamond, b^\diamond) = 3+3-2+3-2+3+3 = 11$
- Another local alignment (g.a. of $a_2, a_3, a_4$ and $b_7, b_8, b_9$)

  $\mathcal{A} = \begin{pmatrix} \texttt{CAT} \\ \texttt{CAT} \end{pmatrix}$, $s(\mathcal{A}) = 3+3+3 = 9$

## Local Alignment Example

- sequences
  $a$ =`GCATTUGCCUU`
  $b$ =`CTTGACCATU`
- similarity $s(x,y) = \begin{cases} -2 & x = - \text{ or } y = - \\ 3 & x = y \\ -1 & x \neq y \end{cases}$

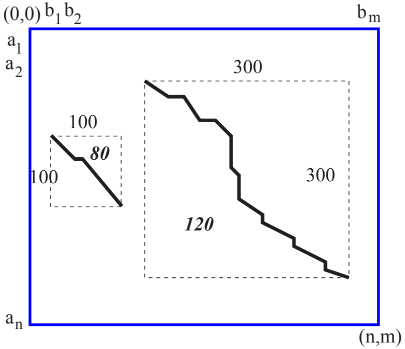
- Local alignment with maximum similarity:
  (global alignment of subsequences $a_4, \dots, a_9$ and $b_2, \dots, b_7$)
  $\begin{array}{l} a^\diamond = \texttt{TTUG-CC} \\ b^\diamond = \texttt{TT-GACC} \end{array}$, $s(a^\diamond, b^\diamond) = 3 + 3 - 2 + 3 - 2 + 3 + 3 = 11$

- Another local alignment (g.a. of $a_2, a_3, a_4$ and $b_7, b_8, b_9$)
  $\mathcal{A} = \begin{pmatrix} \texttt{CAT} \\ \texttt{CAT} \end{pmatrix}$, $s(\mathcal{A}) = 3 + 3 + 3 = 9$

## Local Alignment Example

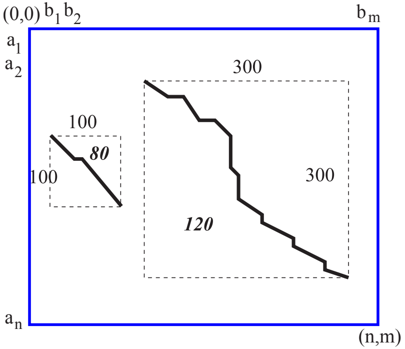
- sequences
  $a = \texttt{GCATTUGCCUU}$
  $b = \texttt{CTTGACCATU}$
- similarity $s(x,y) = \begin{cases} -2 & x = - \text{ or } y = - \\ 3 & x = y \\ -1 & x \neq y \end{cases}$
- Local alignment with maximum similarity:
  (global alignment of subsequences $a_4, \dots, a_9$ and $b_2, \dots, b_7$)
  $$\begin{array}{l} a^\diamond = \texttt{TTUG-CC} \\ b^\diamond = \texttt{TT-GACC} \end{array}, \; s(a^\diamond, b^\diamond) = 3 + 3 - 2 + 3 - 2 + 3 + 3 = 11$$
- Another local alignment (g.a. of $a_2, a_3, a_4$ and $b_7, b_8, b_9$)
  $$\mathcal{A} = \begin{pmatrix} \texttt{CAT} \\ \texttt{CAT} \end{pmatrix}, \; s(\mathcal{A}) = 3 + 3 + 3 = 9$$ Better?

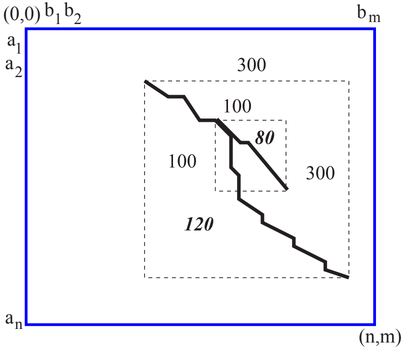
## Drawbacks of “Smith-Waterman” Local Alignment: **Shadowing**



```
longer alignments "shadow" shorter ones
```

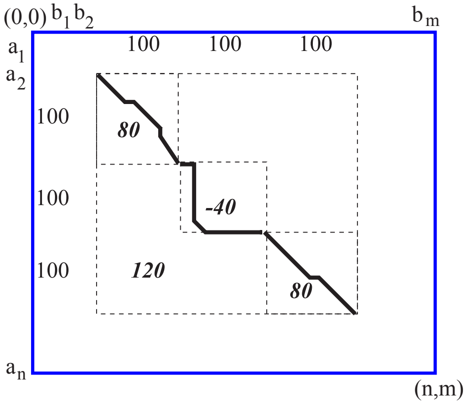
# Drawbacks of "Smith-Waterman" Local Alignment: **Shadowing**





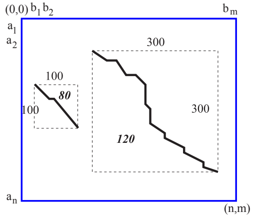
Non-overlapping and overlapping shadowing

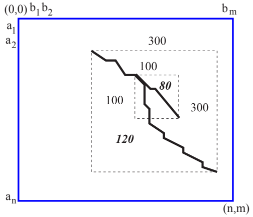
## Drawbacks of “Smith-Waterman” Local Alignment: **Mosaic Effect**

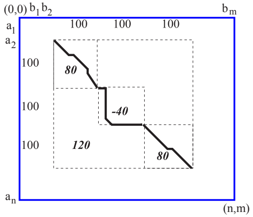


poor "middle parts" hidden in overall scoring

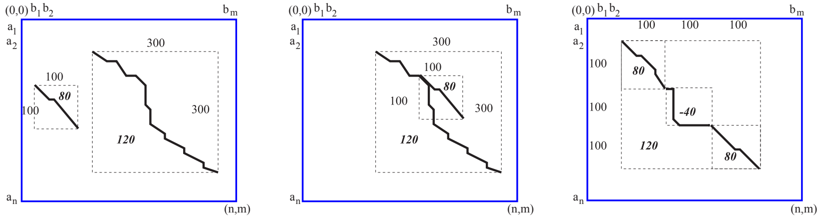
# Overcoming the Drawbacks of Local Alignment





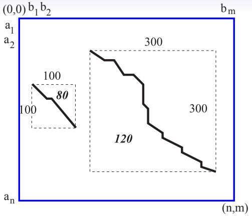


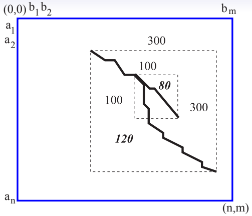
## Overcoming the Drawbacks of Local Alignment

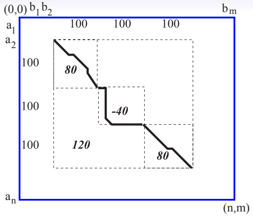


"Smith-Waterman"-similarity does not consider length

## Overcoming the Drawbacks of Local Alignment







"Smith-Waterman"-similarity does not consider length

Better "similarity": $\dfrac{s(\mathcal{A})}{\text{length of subsequences in } \mathcal{A}}$

In example $a=$GCATTUGCCUU, $b=$CTTGACCATU, "shadowing":

$$s\begin{pmatrix}\texttt{TTUG-CC}\\\texttt{TT-GACC}\end{pmatrix} > s\begin{pmatrix}\texttt{CAT}\\\texttt{CAT}\end{pmatrix}$$

but: $$\frac{s\begin{pmatrix}\texttt{TTUG-CC}\\\texttt{TT-GACC}\end{pmatrix}}{|\texttt{TTUGCC}|+|\texttt{TTGACC}|} < \frac{s\begin{pmatrix}\texttt{CAT}\\\texttt{CAT}\end{pmatrix}}{|\texttt{CAT}|+|\texttt{CAT}|}$$

## Overcoming Drawbacks: Normalized Similarity

*Given:*

- sequences $a = a_1 \ldots a_n$ and $b = b_1 \ldots b_m$
- local alignment $\mathcal{A} = (a^\diamond, b^\diamond)$ of $a$ and $b$
  $=$ global alignment of subsequences of $a_i \ldots a_j$ and $b_k \ldots b_l$

*Define:*

- *length of alignment subsequences*

$$[\![\mathcal{A}]\!] := |a_i \ldots a_j| + |b_k \ldots b_l|$$

- *normalized similarity*

$$ns(\mathcal{A}) := \frac{s(\mathcal{A})}{[\![\mathcal{A}]\!] + L}$$

$\boxed{\text{constant } L \text{ controls dependency of } ns(\mathcal{A}) \text{ on length}}$

## Overcoming Drawbacks: Normalized Similarity

*Given:*

- sequences $a = a_1 \dots a_n$ and $b = b_1 \dots b_m$
- local alignment $\mathcal{A} = (a^\diamond, b^\diamond)$ of $a$ and $b$
  $=$ global alignment of subsequences of $a_i \dots a_j$ and $b_k \dots b_l$

*Define:*

- *length of alignment subsequences*

$$[\![\mathcal{A}]\!] := |a_i \dots a_j| + |b_k \dots b_l|$$

- *normalized similarity*

$$ns(\mathcal{A}) := \frac{s(\mathcal{A})}{[\![\mathcal{A}]\!] + L}$$

constant $L$ controls dependency of $ns(\mathcal{A})$ on length

## Overcoming Drawbacks: Normalized Similarity

*Given:*

- sequences $a = a_1 \dots a_n$ and $b = b_1 \dots b_m$
- local alignment $\mathcal{A} = (a^\diamond, b^\diamond)$ of $a$ and $b$
  $=$ global alignment of subsequences of $a_i \dots a_j$ and $b_k \dots b_l$

*Define:*

- *length of alignment subsequences*

$$[\![\mathcal{A}]\!] := |a_i \dots a_j| + |b_k \dots b_l|$$

- *normalized similarity*

$$ns(\mathcal{A}) := \frac{s(\mathcal{A})}{[\![\mathcal{A}]\!] + L}$$

$$\boxed{\text{constant } L \text{ controls dependency of } ns(\mathcal{A}) \text{ on length}}$$

## Effect of constant $L$

| $\mathcal{A}$ | $s(\mathcal{A})$ | $ns(\mathcal{A})$ | | | |
|---|---|---|---|---|---|
| | | $L = 0$ | $L = 1$ | $L = 21$ | $L = 50$ |
| $\begin{pmatrix} \texttt{TTUG-CC} \\ \texttt{TT-GACC} \end{pmatrix}$ | **11** | 0.92 | 0.85 | **0.33** | **0.18** |
| $\begin{pmatrix} \texttt{CAT} \\ \texttt{CAT} \end{pmatrix}$ | 9 | **1.5** | **1.29** | **0.33** | 0.16 |
| $\begin{pmatrix} \texttt{C} \\ \texttt{C} \end{pmatrix}$ | 3 | **1.5** | 1 | 0.14 | 0.06 |

## Normalized Local Alignment

IN: sequences $a = a_1 \dots a_n$ and $b = b_1 \dots b_m$

$$ns(\mathcal{A}) := \frac{s(\mathcal{A})}{[\![\mathcal{A}]\!] + L}$$

OUT: local alignment $\mathcal{A}^*$ of $a$ and $b$ with maximum *normalized similarity* $\lambda^* = ns(\mathcal{A}^*)$

```
no algorithm details today ...
```

## Finding the Best Normalized Local Alignment

1. *Guess* $\lambda^*$
   find best local alignment $\mathcal{A}^*$
   $\lambda^* := ns(\mathcal{A}^*)$

2. *(Try to) improve*: find better $\lambda^*$
   For this purpose, set $\lambda := \lambda^*$ and find new $\lambda^*$ with $\lambda^* - \lambda \geq 0$.
   Note: In one step, we cannot maximize
   $\lambda^* - \lambda = ns(\mathcal{A}) - \lambda = \frac{s(\mathcal{A})}{[\![\mathcal{A}]\!]+L} - \lambda$.
   Thus, maximize $(\lambda^* - \lambda)([\![\mathcal{A}]\!] + L) = s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L)$
   (over all local alignments $\mathcal{A}$; let $\mathcal{A}^*$ be the best alignment).

3. *Iterate* with new estimation $\lambda^* := ns(\mathcal{A}^*)$ until convergence
   (i.e. $\lambda = \lambda^*$).

*Remark:* this algorithm is known as *Dinkelbach's algorithm*

## Finding the Best Normalized Local Alignment

1. *Guess* $\lambda^*$
   find best local alignment $\mathcal{A}^*$
   $\lambda^* := ns(\mathcal{A}^*)$

2. *(Try to) improve*: find better $\lambda^*$
   For this purpose, set $\lambda := \lambda^*$ and find new $\lambda^*$ with $\lambda^* - \lambda \geq 0$.
   Note: In one step, we cannot maximize
   $\lambda^* - \lambda = ns(\mathcal{A}) - \lambda = \frac{s(\mathcal{A})}{[\![\mathcal{A}]\!]+L} - \lambda$.
   Thus, maximize $(\lambda^* - \lambda)([\![\mathcal{A}]\!] + L) = s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L)$
   (over all local alignments $\mathcal{A}$; let $\mathcal{A}^*$ be the best alignment).

3. *Iterate* with new estimation $\lambda^* := ns(\mathcal{A}^*)$ until convergence
   (i.e. $\lambda = \lambda^*$).

*Remark:* this algorithm is known as *Dinkelbach's algorithm*

## Finding the Best Normalized Local Alignment

1. *Guess* $\lambda^*$
   find best local alignment $\mathcal{A}^*$
   $\lambda^* := ns(\mathcal{A}^*)$

2. *(Try to) improve*: find better $\lambda^*$
   For this purpose, set $\lambda := \lambda^*$ and find new $\lambda^*$ with $\lambda^* - \lambda \geq 0$.
   Note: In one step, we cannot maximize
   $\lambda^* - \lambda = ns(\mathcal{A}) - \lambda = \frac{s(\mathcal{A})}{[\![\mathcal{A}]\!]+L} - \lambda$.
   Thus, maximize $(\lambda^* - \lambda)([\![\mathcal{A}]\!] + L) = s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L)$
   (over all local alignments $\mathcal{A}$; let $\mathcal{A}^*$ be the best alignment).

3. *Iterate* with new estimation $\lambda^* := ns(\mathcal{A}^*)$ until convergence
   (i.e. $\lambda = \lambda^*$).

*Remark:* this algorithm is known as *Dinkelbach's algorithm*

# Finding the Best Normalized Local Alignment

1. *Guess* $\lambda^*$
   find best local alignment $\mathcal{A}^*$
   $\lambda^* := ns(\mathcal{A}^*)$

2. *(Try to) improve*: find better $\lambda^*$
   For this purpose, set $\lambda := \lambda^*$ and find new $\lambda^*$ with $\lambda^* - \lambda \geq 0$.
   Note: In one step, we cannot maximize
   $\lambda^* - \lambda = ns(\mathcal{A}) - \lambda = \frac{s(\mathcal{A})}{[\![\mathcal{A}]\!]+L} - \lambda$.
   Thus, maximize $(\lambda^* - \lambda)([\![\mathcal{A}]\!] + L) = s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L)$
   (over all local alignments $\mathcal{A}$; let $\mathcal{A}^*$ be the best alignment).

3. *Iterate* with new estimation $\lambda^* := ns(\mathcal{A}^*)$ until convergence
   (i.e. $\lambda = \lambda^*$).

*Remark:* this algorithm is known as *Dinkelbach's algorithm*

## Dinkelbach's algorithm Converges to Best Normalized Local Alignment

Define $s_\lambda(\mathcal{A}) := s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L)$

**Theorem:**

In step 2 of Dinkelbach's Algorithm,

i) $s_\lambda(\mathcal{A}^*) >= 0$

ii) $\lambda = ns(\mathcal{A})$ is optimum if and only if $s_\lambda(\mathcal{A}^*) = 0$.

This implies $\lambda^*$ converges to best normalized alignment score.

**Proof:** . . .

## Dinkelbach's algorithm Converges to Best Normalized Local Alignment

Define $s_\lambda(\mathcal{A}) := s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L)$

**Theorem:**

In step 2 of Dinkelbach's Algorithm,

i) $s_\lambda(\mathcal{A}^*) >= 0$

ii) $\lambda = ns(\mathcal{A})$ is optimum if and only if $s_\lambda(\mathcal{A}^*) = 0$.

This implies $\lambda^*$ converges to best normalized alignment score.

**Proof:**

i) we know: $\lambda = ns(\mathcal{A})$ for some local alignment $\mathcal{A}$,
therefore: $s_\lambda(\mathcal{A}) = s(\mathcal{A}) - \lambda(\overset{+L}{[\![\mathcal{A}]\!]}) = ([\![\mathcal{A}]\!] + L)(ns(\mathcal{A}) - \lambda)$
$[\![\mathcal{A}]\!] + L > 0; ns(\mathcal{A}) = \lambda \Rightarrow ns(\mathcal{A}) - \lambda = 0$ Since $s_\lambda(\mathcal{A}) = 0$,
$\max_{\mathcal{A}} s_\lambda(\mathcal{A}) \geq 0$ $\square$

## Dinkelbach's algorithm Converges to Best Normalized Local Alignment

Define $s_\lambda(\mathcal{A}) := s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L)$

**Theorem:**

In step 2 of Dinkelbach's Algorithm,

i) $s_\lambda(\mathcal{A}^*) >= 0$

ii) $\lambda = ns(\mathcal{A})$ is optimum if and only if $s_\lambda(\mathcal{A}^*) = 0$.

This implies $\lambda^*$ converges to best normalized alignment score.

**Proof:**

ii) "$\Longrightarrow$": $s_\lambda(\mathcal{A}^*) = ([\![\mathcal{A}^*]\!] + L)(ns(\mathcal{A}^*) - \lambda) \leq 0$, since $[\![\mathcal{A}^*]\!] + L > 0$ and $ns(\mathcal{A}^*) - \lambda \leq 0$. With (i): $s_\lambda(\mathcal{A}^*) = 0$.

"$\Longleftarrow$": Let $\max_\mathcal{A} s_\lambda(\mathcal{A}) = 0$. Assume $\mathcal{A}$ is not optimal, i.e. there is an $\mathcal{A}'$ with $ns(\mathcal{A}') > \lambda$, then $s_\lambda(\mathcal{A}') = ([\![\mathcal{A}']\!] + L)(ns(\mathcal{A}') - \lambda) > 0$ , since $[\![\mathcal{A}']\!] + L > 0$ and $ns(\mathcal{A}') - \lambda > 0$, contradicting $\max_\mathcal{A} s_\lambda(\mathcal{A}) = 0$. $\square$

## How to maximize $s_\lambda(\mathcal{A}) := s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L)$?

Maximize $s_\lambda(\mathcal{A})$ by solving an instance of local alignment:

$$\begin{aligned}
& s_\lambda(\mathcal{A}) \text{ maximum} \\
\equiv\ & s(\mathcal{A}) - \lambda([\![\mathcal{A}]\!] + L) \text{ maximum} \\
\equiv\ & s(\mathcal{A}) - \lambda[\![\mathcal{A}]\!] - \lambda L \text{ maximum} \\
\equiv\ & s(\mathcal{A}) - \lambda[\![\mathcal{A}]\!] \text{ maximum} \\
\equiv\ & \sum_k s(a_k^\diamond, b_k^\diamond) - \lambda[\![\mathcal{A}]\!] \text{ maximum} \\
\equiv\ & \sum_k (s(a_k^\diamond, b_k^\diamond) - \{\lambda \text{ if } a_k^\diamond \neq -\} - \{\lambda \text{ if } b_k^\diamond \neq -\}) \text{ maximum} \\
\equiv\ & s'(\mathcal{A}) \text{ maximum, where } s' \text{ uses modified base similarity}
\end{aligned}$$

$$s'(x,y) = \begin{cases} s(x,y) - 2\lambda & x, y \in \Sigma \\ s(x,y) - \lambda & x = - \text{ or } y = - \end{cases}$$

## Example

$a =$GCATTUGCCUU and $b =$CTTGACCATU

$s(x, y) = \{3 \text{ match}; -1 \text{ mismatch}; -2 \text{ gap}\}$; L:=10

1. $\mathcal{A}^* = \begin{pmatrix}\texttt{TTUG-CC}\\ \texttt{TT-GACC}\end{pmatrix}$, $s(\mathcal{A}^*) = 11$, $ns(\mathcal{A}^*) = 11/22 = 0.5 \to \lambda$
2. solve local alignment of $a$ and $b$ for modified similarity $s'(x, y) = \{2 \text{ match}; -2 \text{ mismatch}; -2.5 \text{ gap}\}$

   $\Rightarrow$ new $\mathcal{A}^* = \begin{pmatrix}\texttt{CAT}\\ \texttt{CAT}\end{pmatrix}$, $s_\lambda(\mathcal{A}^*) = 9 - 16\lambda = 1$,

   $s(\mathcal{A}^*) = 9$, $ns(\mathcal{A}^*) = 9/16 = 0.5625 \to \lambda$
3. solve local alignment of $a$ and $b$ for new modified similarity $s'(x, y) = \{1.875 \text{ match}; -2.125 \text{ mismatch}; -2.5625 \text{ gap}\}$

   $\Rightarrow$ new $\mathcal{A}^* = \begin{pmatrix}\texttt{CAT}\\ \texttt{CAT}\end{pmatrix}$, $s_\lambda(\mathcal{A}^*) = 0$,

   $s(\mathcal{A}^*) = 9$, $ns(\mathcal{A}^*) = 9/16 = 0.5625$
4. convergence $\Rightarrow \lambda^* = 0.5625$.

## Example

$a =$ GCATTUGCCUU and $b =$ CTTGACCATU

$s(x, y) = \{3 \text{ match}; -1 \text{ mismatch}; -2 \text{ gap}\}$; L:=10

1. $\mathcal{A}^* = \begin{pmatrix} \texttt{TTUG-CC} \\ \texttt{TT-GACC} \end{pmatrix}$, $s(\mathcal{A}^*) = 11$, $ns(\mathcal{A}^*) = 11/22 = 0.5 \rightarrow \lambda$
2. solve local alignment of $a$ and $b$ for modified similarity
   $s'(x, y) = \{2 \text{ match}; -2 \text{ mismatch}; -2.5 \text{ gap}\}$

   $\Rightarrow$ new $\mathcal{A}^* = \begin{pmatrix} \texttt{CAT} \\ \texttt{CAT} \end{pmatrix}$, $s_\lambda(\mathcal{A}^*) = 9 - 16\lambda = 1$,

   $s(\mathcal{A}^*) = 9$, $ns(\mathcal{A}^*) = 9/16 = 0.5625 \rightarrow \lambda$
3. solve local alignment of $a$ and $b$ for new modified similarity
   $s'(x, y) = \{1.875 \text{ match}; -2.125 \text{ mismatch}; -2.5625 \text{ gap}\}$

   $\Rightarrow$ new $\mathcal{A}^* = \begin{pmatrix} \texttt{CAT} \\ \texttt{CAT} \end{pmatrix}$, $s_\lambda(\mathcal{A}^*) = 0$,

   $s(\mathcal{A}^*) = 9$, $ns(\mathcal{A}^*) = 9/16 = 0.5625$
4. convergence $\Rightarrow \lambda^* = 0.5625$.

## Example

$a = \texttt{GCATTUGCCUU}$ and $b = \texttt{CTTGACCATU}$
$s(x, y) = \{3 \text{ match}; -1 \text{ mismatch}; -2 \text{ gap}\}$; L:=10

1. $\mathcal{A}^* = \begin{pmatrix}\texttt{TTUG-CC}\\ \texttt{TT-GACC}\end{pmatrix}$, $s(\mathcal{A}^*) = 11$, $ns(\mathcal{A}^*) = 11/22 = 0.5 \to \lambda$
2. solve local alignment of $a$ and $b$ for modified similarity
   $s'(x, y) = \{2 \text{ match}; -2 \text{ mismatch}; -2.5 \text{ gap}\}$

   $\Rightarrow$ new $\mathcal{A}^* = \begin{pmatrix}\texttt{CAT}\\ \texttt{CAT}\end{pmatrix}$, $s_\lambda(\mathcal{A}^*) = 9 - 16\lambda = 1$,

   $s(\mathcal{A}^*) = 9$, $ns(\mathcal{A}^*) = 9/16 = 0.5625 \to \lambda$
3. solve local alignment of $a$ and $b$ for new modified similarity
   $s'(x, y) = \{1.875 \text{ match}; -2.125 \text{ mismatch}; -2.5625 \text{ gap}\}$

   $\Rightarrow$ new $\mathcal{A}^* = \begin{pmatrix}\texttt{CAT}\\ \texttt{CAT}\end{pmatrix}$, $s_\lambda(\mathcal{A}^*) = 0$,

   $s(\mathcal{A}^*) = 9$, $ns(\mathcal{A}^*) = 9/16 = 0.5625$
4. convergence $\Rightarrow \lambda^* = 0.5625$.

## Example

$a$ =GCATTUGCCUU and $b$ =CTTGACCATU

$s(x, y) = \{3 \text{ match}; -1 \text{ mismatch}; -2 \text{ gap}\}$; L:=10

1. $\mathcal{A}^* = \begin{pmatrix}\texttt{TTUG-CC}\\ \texttt{TT-GACC}\end{pmatrix}$, $s(\mathcal{A}^*) = 11$, $ns(\mathcal{A}^*) = 11/22 = 0.5 \rightarrow \lambda$
2. solve local alignment of $a$ and $b$ for modified similarity $s'(x, y) = \{2 \text{ match}; -2 \text{ mismatch}; -2.5 \text{ gap}\}$

   $\Rightarrow$ new $\mathcal{A}^* = \begin{pmatrix}\texttt{CAT}\\ \texttt{CAT}\end{pmatrix}$, $s_\lambda(\mathcal{A}^*) = 9 - 16\lambda = 1$,

   $s(\mathcal{A}^*) = 9$, $ns(\mathcal{A}^*) = 9/16 = 0.5625 \rightarrow \lambda$
3. solve local alignment of $a$ and $b$ for new modified similarity $s'(x, y) = \{1.875 \text{ match}; -2.125 \text{ mismatch}; -2.5625 \text{ gap}\}$

   $\Rightarrow$ new $\mathcal{A}^* = \begin{pmatrix}\texttt{CAT}\\ \texttt{CAT}\end{pmatrix}$, $s_\lambda(\mathcal{A}^*) = 0$,

   $s(\mathcal{A}^*) = 9$, $ns(\mathcal{A}^*) = 9/16 = 0.5625$
4. convergence $\Rightarrow \lambda^* = 0.5625$.

## Conclusion: Normalized Alignment

- Normalized alignment helps to eliminate shadowing and mosaic effect
- General technique: Dinkelbach's Algorithm solves "Fractional programming" problems, where objective function is fraction
- Dinkelbach requires to solve parametric optimization problem (for $s_\lambda(\mathcal{A})$); works by distributing length-dependent terms
- Usually few iterations, but no worst-case bound; other methods are available (see Arslan, Bioinformatics, 2001)
- Normalized local alignment only a few times slower than local alignment
- Extensions to more complex scoring, e.g. affine gap cost