3 Comparing Sequences

13 November 2025

Prof. Dr. Sebastian Wild

CS594 (Winter 2025-26)
Philipps-Universität Marburg
version `2025-11-20 22:38` H

Outline

3 Comparing Sequences

3.1 Sequence Alignment

Sequence Similarity

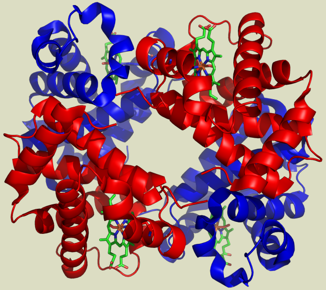
Example: two proteins from *human hemoglobin*

Human Hemoglobin α globin subunit https://www.uniprot.org/uniprotkb/P69905

Human Hemoglobin β globin subunit https://www.uniprot.org/uniprotkb/P68871

⇝ *essentially symmetric copies with same function*

3D Structure of hemoglobin



https://commons.wikimedia.org/wiki/File:1GZX_Haemoglobin.png

Sequences of the subunits (142 resp. 147 amino acids):

```
MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR

MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
```

These are supposed to be "similar"!?

Alignment by EMBOSS Needle https://www.ebi.ac.uk/jdispatcher/psa

```
MV-LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-----HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR
|| |:| :|: | | ||||  :  | | ||| |: : :| |: :|  | |||      |: :|| |||||  | :: :||:|::    : ||:||  || ||| ||:|| : |:  || |   |||| | |:  | :| |:  |  ||
MVHLTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
```

| = same amino acid (65x); : = similar amino acids (25x) ⇝ 60% same

String Distances

Mutations mean much in bioinformatics needs fuzzy comparisons . . .
How can we formally define these?

- This unit studies wide class of options
- Algorithmically, all are similar to deal with
- Unfortunately, general case again hard . . .

- Simplest string distance function: ***Hamming distance*** d_H = #mismatches
  - ϟ only defined for strings of same length
  - How about strings like this:

    A = `alongsharedstring`
    B = `longsharedstrings` $\rightsquigarrow$ $d_H(A, B) = |A| = 17$ *These are maximally different!?*

$\rightsquigarrow$ *Need a more flexible notion . . .*

Edit Distance

Natural idea for distances: describe **how** to get from A to B $\rightsquigarrow$ *relative compression!*

$A[0..17) =$ `alongsharedstring`
$B[0..17) =$ `longsharedstrings`

"Edit script":

0. Start with S_1.

1. Delete $S_1[0]$

2. Insert `s` at end of S_1.

$\rightsquigarrow$ 2 character operations needed $\rightsquigarrow$ $d_{\text{edit}}(A, B) = 2$

Edit Distance Problem

- **Given:** String $A[0..m)$ and $B[0..n)$ over alphabet $\Sigma = [0..\sigma)$.
- **Goal:** $d_{\text{edit}}(A, B) =$ minimal # symbol operations to transform A into B
  operations can be insertion/deletion/substitution of single character

  + optimal edit script (with this number of operations)

Edit Distance Example

Example: edit distance $d_{\text{edit}}(A, B)$ with $A = \texttt{algorithm}$, $B = \texttt{logarithm}$?

```
012345678
algorithm
logarithm
```

Edit script:

1. Delete $A[0]$
2. Insert `o` after $A[1] = \texttt{l}$
3. Replace $A[3] = \texttt{o}$ by `a`

Compact representation of edit script: *String alignment*

```
0123456789
al-gorithm
-|+|x|||||
-logarithm
```

Formally: string over pairs of letters or *gap symbols*

$$\left\{\begin{bmatrix}c\\c\end{bmatrix} : c \in \Sigma\right\} \cup \left\{\begin{bmatrix}c\\-\end{bmatrix}, \begin{bmatrix}-\\c\end{bmatrix} : c \in \Sigma\right\} \cup \left\{\begin{bmatrix}c\\c'\end{bmatrix} : c, c' \in \Sigma, c \neq c'\right\}$$

$\rightsquigarrow$ Edit distance $= \#\ \left[\begin{smallmatrix}c\\-\end{smallmatrix}\right], \left[\begin{smallmatrix}-\\c\end{smallmatrix}\right], \left[\begin{smallmatrix}c\\c'\end{smallmatrix}\right]$ with $c \neq c'$

Edit Distance and Longest Common Subsequence

- Note: close relation to *longest common subsequence*
  Optimal edit script $\approx$ maximal number of matches $=$ longest common subsequence
- But: Optimal alignment may not contain any longest common subsequence

```
axxa  axxa  axxa
|  |  |  |  |  |
a  ayya  ayya  ayy


 axxaaxxaaxxa
 |  ||  ||
aayyaayyaayy
```

- LCS and edit distance are equivalent if we only allow insert and delete operations

3.2 Dynamic Programming

Recap: The 6 Steps of Dynamic Programming

$\rightsquigarrow$ see *Efficient Algorithms*

1. Define **subproblems** (and relate to original problem)
2. **Guess** (part of solution) $\rightsquigarrow$ local brute force
3. Set up **DP recurrence** (for quality of solution)
4. Recursive implementation with **Memoization**
5. Bottom-up **table filling** (topological sort of subproblem dependency graph)
6. **Backtracing** to reconstruct optimal solution

- Steps 1–3 require insight / creativity / intuition;
  Steps 4–6 are mostly automatic / same each time

$\rightsquigarrow$ Correctness proof usually at level of DP recurrence

👍 running time too! worst case time = #subproblems · time to find single best guess

Edit Distance by DP

1. **Subproblems:** (i,j) for $0 \le i \le m$, $0 \le j \le m$ compute $d_{\text{edit}}(A[0..i), B[0..j))$
2. **Guess:** What to do with last positions? (insert/delete/(mis)match)
3. **Recurrence:** $D(i,j) = d_{\text{edit}}(A[0..i), B[0..j))$

$$D(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{cases} D(i-1,j)+1, \\ D(i,j-1)+1, \\ D(i-1,j-1) + \big[A[i-1] \ne B[j-1]\big] \end{cases} & \text{otherwise} \end{cases}$$

$\rightsquigarrow$ $O(nm)$ subproblems

- $O(1)$ time to check all guesses (per subproblem)

$\rightsquigarrow$ $O(nm)$ overall time and space

- An optimal *edit script* can be constructed by a *backtrace* (see below)

Edit Distance – Step 4: Memoization

- Write **recursive** function to compute recurrence
- But *memoize* all results! (symbol table: subproblem $\mapsto$ optimal cost)

$\rightsquigarrow$ First action of function: check if subproblem known

- If so, return cached optimal cost
- Otherwise, compute optimal cost and remember it!

1. Subproblems
2. Guess!
3. DP Recurrence
4. Memoization
5. Table Filling
6. Backtrace

```
procedure editDist($i$, $j$):
    if $i == 0$
        return $j$
    else if $j == 0$
        return $i$
    end if
    $best := +\infty$
    $D_i := \text{cachedED}(i, j-1) + 1$
    $D_d := \text{cachedED}(i-1, j) + 1$
    $D_m := \text{cachedED}(i-1, j-1) + [A[i] \neq B[j]]$
    $best := \min\{D_d, D_i, D_m\}$
    return $best$
```

$$D(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{cases} D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + [A[i-1] \neq B[j-1]] \end{cases} & \text{otherwise} \end{cases}$$

```
procedure cachedED($r[i..j)$, $c[i..j)$):
    // $D[0..m][0..n]$ initialized to NULL at start
    if $D[i][j]$ == NULL
        $D[i][j] := \text{editDist}(i, j)$
    return $D[i][j]$
```

Edit Distance – Step 5: Table Filling

- Recurrence induces a DAG on subproblems (who calls whom)
  - Memoized recurrence traverses this DAG (DFS!)
  - We can slightly improve performance by systematically computing subproblems following a fixed topological order

1. Subproblems
2. Guess!
3. DP Recurrence
4. Memoization
5. Table Filling
6. Backtrace

- **Topological order** here: lexicographic by (i, j)

```
procedure editDist($A[0..m]$, $B[0..n]$):
    $D[0..m][0..n]$ := new array
    for $i = 0, 1, \dots, m$ // iterate over subproblems ...
        for $j = 0, 1 \dots, n$ // ... in topological order
            if $i == 0$
                $D[i][j] := j$
            else if $j == 0$
                $D[i][j] := i$
            else
                $D[i][j] := \min \begin{cases} D[i][j-1] + 1, \\ D[i-1][j] + 1, \\ D[i-1][j-1] + [A[i-1] \neq B[j-1]] \end{cases}$
    return $D[m][n]$
```

- Same Θ-class as memoized recursive function
- In practice usually substantially faster
  - lower overhead
  - predictable memory accesses

Edit Distance – Step 6: Backtracing

- So far, only determine the **cost** of an optimal solution
  - But we also want the solution itself
- By *retracing* our steps, we can construct optimal edit script

```
procedure editScript($A[0..m)$, $B[0..n)$):
    $D[0..m)[0..n)$ := editDist($A[0..m)$, $B[0..n)$)
    return traceback($m$, $n$)

procedure traceback($i$, $j$):
    if $i == 0$
        return Insert($B[0]$), ..., Insert($B[j-1]$)
    else if $j == 0$
        return Delete($A[0]$), ..., Delete($A[i-1]$)
    else if $D[i][j] == D[i][j-1] + 1$
        return traceback($i$, $j-1$), Insert($B[j-1]$)
    else if $D[i][j] == D[i-1][j] + 1$
        return traceback($i-1$, $j$), Delete($B[i-1]$)
    else if $A[i-1] == B[j-1]$
        return traceback($i-1$, $j-1$)
    else return traceback($i-1$, $j-1$), Replace($A[i-1] \to B[j-1]$)
```

1. Subproblems
2. Guess!
3. DP Recurrence
4. Memoization
5. Table Filling
6. Backtrace

- follow recurrence a second time
- always have for running time: backtracing = O(computing M)

$\rightsquigarrow$ computing optimal cost and computing optimal solution have same complexity

3.3 Global – Local – Semilocal

Local Alignment

So far, we assumed that we know similar regions.
How to detect significantly similar regions hidden in larger strings?

$\rightsquigarrow$ Allow new edit script operations (all cost 0):

- IgnorePrefix($A[0..i)$) free deletes at beginning
- IgnorePrefix($B[0..j)$) free inserts at beginning
- IgnoreSuffix($A[i..m)$) free deletes at end
- IgnoreSuffix($B[j..n)$) free inserts at end

$\rightsquigarrow$ *Local Alignment*

- Easy to incorporate in DP recurrence:

  0. switch to **maximizing score** (instead min difference), otherwise empty substring is best

  $\rightsquigarrow$ Matches contribute $+1$ reward, rest penalty (negative score)

  1. Always allow 4th option: **start** a **new** local alignment from here (at score 0)

  2. Allow to finish at any $D[i][j]$ $\rightsquigarrow$ free suffix

Local Alignment Recurrence

$$D(i,j) = \begin{cases} \mathbf{0} & \text{if } j = 0 \\ \mathbf{0} & \text{if } i = 0 \\ \min \begin{cases} \mathbf{0}, \\ D(i-1,j) \mathbf{-1}, \\ D(i,j-1) \mathbf{-1}, \\ D(i-1,j-1) + \left[A[i-1] \mathbf{=} B[j-1]\right] \end{cases} & \text{otherwise} \end{cases}$$

Optimal local alignment score: $\displaystyle\max_{i\in[0..m],j\in[0..n]} D[i][j]$

Semilocal Aligment a.k.a. Fitting Alignment

Slight twist: We know conserved region, but need to find best match in larger sequence.
What substring of $B[0..n)$ is the best match for $A[0..m)$? (typically then $m \ll n$)

$\rightsquigarrow$ only allow IgnorePrefix($B[0..j)$) and IgnoreSuffix($B[j..n)$)

$$\rightsquigarrow\ D(i,j) = \begin{cases} -i & \text{if } j = 0 \\ \mathbf{0} & \text{if } i = 0 \\ \min \begin{cases} D(i-1,j) \mathbf{- 1}, \\ D(i,j-1) \mathbf{- 1}, \\ D(i-1,j-1) + \big[A[i-1] \mathbf{=} B[j-1]\big] \end{cases} & \text{otherwise} \end{cases}$$

Optimal local alignment score: $\max_{j \in [0..n]} D[m][j]$

3.4 General Scores & Affine Gap Costs

General Scores

DP algorithm remains unchanged if we let contribution of (mis)match $A[i-1]$ vs $B[j-1]$ depend on used letters.

- For example, replacing amino acid with chemically similar one might not affect function $\rightsquigarrow$ contributes small positive score
- replacing amino acid with dissimilar one $\rightsquigarrow$ negative score

Formally, any function giving additive scores for columns $S : (\Sigma \cup \{-\})^2 \setminus \left\{\left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]\right\} \to \mathbb{R}$ works.

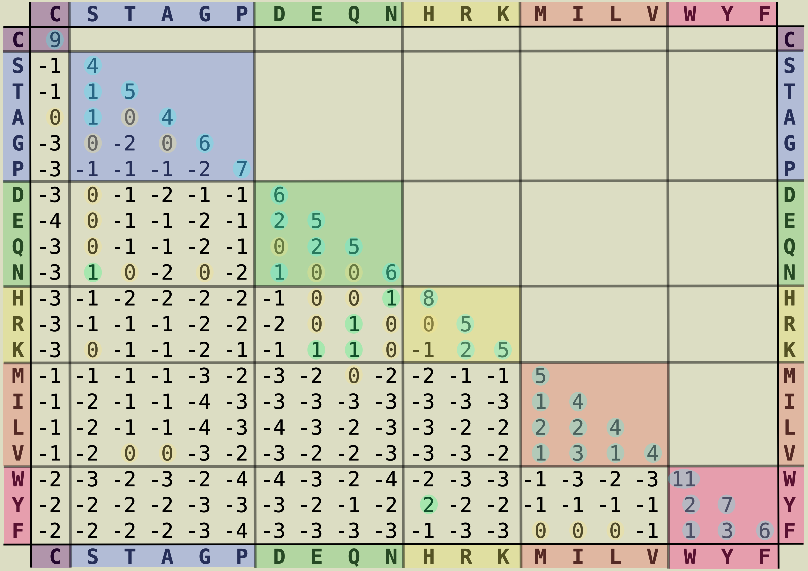
General Alignment Score S:

- symmetric matches/substitutions matrix $p : \Sigma \times \Sigma \to \mathbb{R}$ $\qquad (p(a,b) = p(b,a))$
- gap penalty $g \in \mathbb{R}$

$\rightsquigarrow$ $S\left(\left[\begin{smallmatrix}c\\c'\end{smallmatrix}\right]\right) = p(a,b),\ S\left(\left[\begin{smallmatrix}c\\-\end{smallmatrix}\right]\right) = S\left(\left[\begin{smallmatrix}-\\c\end{smallmatrix}\right]\right) = g$

$\rightsquigarrow$ score of alignment sum of scores of columns

BLOSOM Matrices



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

https://en.wikipedia.org/wiki/BLOSUM#/media/File:Blosum62-dayhoff-ordering.svg

Affine Gap costs

In sequence evolution, insertions of single stretch of k characters much more likely than k isolated (single-character) insertions
So far, we score these the same.

$\rightsquigarrow$ *affine gap costs:*
score k contiguous insertions (or k contiguous deletions) instead as $g_0 + k \cdot g$
(usually then $g_0 \gg g$)

- If we represent contiguous insertions as $\begin{bmatrix}\vdash\\c_1\end{bmatrix}\begin{bmatrix}-\\c_2\end{bmatrix}\cdots\begin{bmatrix}-\\c_k\end{bmatrix}$
can assign $S(\begin{bmatrix}\vdash\\c\end{bmatrix}) = g_0 + g$ and $S\begin{bmatrix}-\\c\end{bmatrix} = g$.
- DP algorithm can be extended to handle these refined scores
  $\rightsquigarrow$ exercises

3.5 Bounded-Distance Alignments

Good Alignment or Abort

3.6 Exhaustive Tabulation

Four Russians?

The *exhaustive-tabulation technique* to follow is often called "Four Russians trick" . . .

- The algorithmic technique was published 1970 by
  V. L. Arlazarov, E. A. Dinitz, M. A. Kronrod, and I. A. Faradžev
- all worked in Moscow at that time . . . but not even clear if all are Russians

  (Arlazarov and Kronrod are Russian)
- American authors coined the othering term "Method of Four Russians"
  . . . name in widespread use

A Trick for Matrix Multiplication

Suppose we want to multiply two $n \times n$ Boolean matrices $C = A \cdot B$.

We divide A, B, and C into $\ell \times \ell$ *micro matrices*.

$\rightsquigarrow$ C consists of $\left(\frac{n}{\ell}\right)^2$ micro matrices, each of which is the sum of $\frac{n}{\ell}$ micro-matrix products.

The number of *different* possible micro matrix products is $L = 2^{\ell^2} \cdot 2^{\ell^2}$.
If we pick $\ell = \frac{1}{4}\sqrt{\lg n}$, we have only $L = 2^{2\ell^2} = \sqrt{n}$ different products.

$\rightsquigarrow$ ***Exhaustive Tabulation:*** Can *precompute* all $\sqrt{n}$ *possible* micro-matrix sums/products!

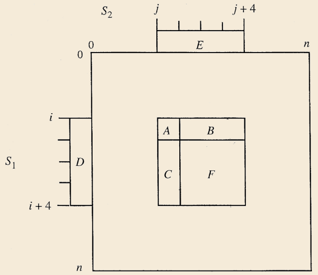
For two micro matrices a and b, we store $a \cdot b$ at the offset $a_{1,1} \ldots a_{\ell,\ell} b_{1,1} \ldots b_{\ell,\ell}$, where we interpret this bitstring as a binary number.
On a word RAM, we can use this as indirect memory access in $O(1)$ time.

$\rightsquigarrow$ Any micro matrix sum/product takes $O(1)$ time
after a total of $O(\sqrt{n} \cdot \log^{3/2} n)$ preprocessing.

The total time to compute one micro matrix in C is thus $O(\frac{n}{\ell})$.
So the total time to compute C is $O(n^3/\ell^3) = O(n^3/\log^{3/2} n)$.

Note: By taking $n \times \ell$ resp. $\ell \times n$ *"micro strips"* instead of squares, we can choose
$\ell = \Theta(\log n)$ and obtain final time $O(n^3/\log^2 n)$.

Exhaustive Tabulation for Edit Distance



Gusfield, *Algorithms on Strings, Trees, and Sequences*, Fig. 12.21

Micro matrix

- Split $D(i,j)$ matrix Again $\ell \times \ell$ submatrices corresponding to ℓ-char substrings of S_1 and S_2
- values in F only depend on A, B, C, D, and E!

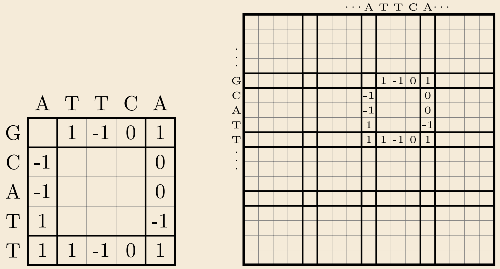
$\rightsquigarrow$ can make progress micro matrix by micro matrix

But . . . exhaustive tabulation doesn't seem to work! *The values of* $D(i,j)$ *keep increasing!*
How shall we bound the number of possible micro matrices?

- **Observation:** The difference between neighboring cells $D(i,j)$ and $D(i,j+1)$ respectively $D(i,j)$ and $D(i+1,j)$ is in $\{\mathbf{-1}, \mathbf{0}, \mathbf{+1}\}$.
  - $D(i,j+1) \le D(i,j)+1$ is trivial from recurrence
  - $D(i,j) \le D(i,j+1)+1$ needs closer look / case distinction

$\rightsquigarrow$ Apply tabulation for offset, not actual values in $D(i,j)$

Putting the Micro Matrices together



Brubach Ghurye, *A Succinct Four Russians Speedup for Edit Distance Computation and One-against-many Banded Alignment*, Fig. 1

- Choose micro matrices with one row/col overlapping
- initialize first row and col (as per recurrence)
- number of different micro matrices: $\le \sigma^{2\ell} \cdot 3^{2(\ell-1)}$

$\rightsquigarrow$ $\ell \le \frac{1}{4}\log_{3\sigma}(n)$ for $O(\sqrt{n})$ micro matrices

- For constant σ, $\ell = \Theta(\log n)$ and we have to fill n^2/ℓ^2 micro matrices
- Filling table cells not needed; grid row/col only fed into next lookup table

$\rightsquigarrow$ $O(1)$ time per micro matrix

$\rightsquigarrow$ $O(n^2/\log^2 n)$ time overall

Can we do better?

Theorem 3.1 (Conditional Lower Bound for Edit Distance)
An algorithm for computing the edit distance of any two strings of length n in time $O(n^{2-\delta})$ for constant $\delta > 0$ would refute the Strong Exponential-Time Hypothesis. ◀

Backurs, Indyk: *Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)*, STOC **2015**

Definition 3.2 (Exponential-Time Hypothesis)
The *Exponential-Time Hypothesis (ETH)* asserts that there is a constant $\delta > 0$ so that every algorithm for 3SAT requires $\Omega(2^{\delta k})$ time, where k is the number of variables. ◀

Definition 3.3 (Strong Exponential-Time Hypothesis)
The *Strong Exponential-Time Hypothesis (SETH)* asserts that for every $\varepsilon > 0$ there is a k such that kSAT requires $\Omega(2^{(1-\varepsilon)k})$ time, where k is the number of variables. ◀

Unlikely to see "truly subquadratic" algorithms *(even for constant alphabets)*

3.7 Linear-Space Alignments

Saving Space is Easy for Score

Assume here that $n \le m$.

DP for $D[i][j]$,
only need $O(n)$ space:

- $D[i][j]$ depends on $D[i-1][j]$, $D[i][j-1]$, and $D[i-1][j-1]$.
- clearly enough to keep previous and current row of D
- actually, can even *overwrite* as we go along
  $\rightsquigarrow$ single row sufficient

```
procedure Score($A[0..m)$, $B[0..n)$)
    $D$ := ScoresRow($A$, $B$)
    return $D[n]$

procedure ScoresRow($A[0..m)$, $B[0..n)$)
    $D[0..n]$ := new array
    for $j$ := $0, \ldots, n$
        $D[j]$ := $j \cdot g$
    for $i$ := $1, \ldots, m$
        $match$ := $(i-1) \cdot g$
        for $j = 1, \ldots, n$
            $new$ := $\min \begin{cases} match + p(A[i-1], B[j-1]) \\ D[j] + g \\ D[j-1] + g \end{cases}$
            $match$ := $D[j]$
            $D[j]$ := $new$
```

The Middle-Point Problem

*To reconstruct alignment/edit script **using standard backtrace**, need full table* $D[0..n][0..m]$.

But can also reconstruct edit script using Divide & Conquer DP approach!

- **Idea:** Construct edit script for turning $A[0..m/2)$ into $B[0..j^*)$
  and for turning $A[m/2..m)$ into $B[j^*..n)$
- But we don't know *middle point* j^* ... so need to **guess** it! $\rightsquigarrow$ use DP!

Hold on, are we running in circles?

No! j^* *optimizes **sum of scores** of* $A[0..m/2) \to B[0..j^*)$ *and* $A[m/2..m) \to B[j^*..n)$
$\rightsquigarrow$ *Can use linear-space ScoresRow!*

- Score for $A[0..m/2) \longrightarrow B[0..j^*)$ is $D[m/2][j^*]$
- For $A[m/2..m) \longrightarrow B[j^*..n)$ we don't have an entry in D!
- But we can **reverse** A and B

Linear-Space Alignment

```
procedure editScript($A[0..m)$, $B[0..n)$)
    if $m == 0$ then return Insert($B[0]$), ..., Insert($B[n-1]$)
    else if $n == 0$ then return Delete($A[0]$), ..., Delete($A[m-1]$)
    else if $m == 1$
        $j := \arg\min_{0 \le j < n} p(A[0], B[j])$
        return Insert($B[0..j)$), Replace($A[0]$, $B[j]$), Insert($B[j+1..n)$)
    else
        $i^* := \lfloor \frac{m}{2} \rfloor$
        $D_{top}$    := ScoresRow($A[0..i^*)$, $B$)
        $D_{bottom}$ := ScoresRow($A[i^*..m)^R$, $B^R$) // $s^R$ is $s$ reversed
        $j^* := \arg\min_{0 \le j \le n} D_{top}[j] + D_{bottom}[n-j]$
        return editScript($A[0..i^*)$, $B[0..j^*)$), editScript($A[i^*..m)$, $B[j^*..n)$)
    endif
```

- Non-recursive cost $\Theta(n \cdot m)$ for ScoresRow
- "Area" $n \cdot m$ in recursive calls is **halved** in each step.

$\rightsquigarrow$ Total time $\Theta(nm)$, but using only $\Theta(\min n, m)$ space

3.8 Multiple Sequence Alignment

Multiple-Sequence Alignment

Biological sequences are often too noisy to recognize preserved regions from pairwise alignments.

A shared region between two sequences could be random coincidence.
A shared region between **many** sequences hardly are.

"One or two homologous sequences whisper . . . a full multiple alignment shouts out loud"
(Arthus Lesk)

Example: *β-globin* in different species:

```
Xenopus     MVHWTAEEKAAITSVWQKVNVEHDGHDALGRLLIVYPWTQRYFSNFGNLSNSAAVAGNAKVQAHGKKVLSAVGNAISHIDSVKSSLQQLSKIHATELFVDPENFKRFGGVLVIVLGAKLGT-AFTPKVQAAWEKFIAVLVDGLSQGYN
Zebrafish   MVEWTDAERTAILGLWGKLNIDEIGPQALSRCLIVYPWTQRYFATFGNLSSPAAIMGNPKVAAHGRTVMGGLERAIKNMDNVKNTYAALSVMHSEKLHVDPDNFRLLADCITVCAAMKFGQAGFNADVQEAWQKFLAVVVSALCRQYH
Chicken     MVHWTAEEKQLITGLWGKVNVAECGAEALARLLIVYPWTQRFFASFGNLSSPTAILGNPMVRAHGKKVLTSFGDAVKNLDNIKNTFSQLSELHCDKLHVDPENFRLLGDILIIVLAAHFSK-DFTPECQAAWQKLVRVVAHALARKYH
Human       MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGK-EFTPPVQAAYQKVVAGVANALAHKYH
Mouse       MVHLTDAEKAAVSCLWGKVNSDEVGGEALGRLLVVYPWTQRYFDSFGDLSSASAIMGNAKVKAHGKKVITAFNDGLNHLDSLKGTFASLSELHCDKLHVDPENFRLLGNMIVIVLGHHLGK-DFTPAAQAAFQKVVAGVATALAHKYH
            **. *  *:  :  :* *:*  . * :**.* *:*******:* .**:**.  *: **  * ***:.*: ..  .: ::*.:*.:   ** :*. :*.***:**: :.. :    . ::.   *.   * *::*.:  :. .*.: *:
```

African Clawed Frog (Xenopus laevis): P02133
Zebrafish (Danio rerio): Q90486
Chicken (Gallus gallus): P02112
Human (Homo sapiens): P68871
Mouse (Mus musculus): P02088

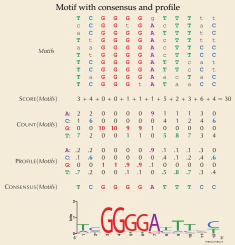
https://www.ebi.ac.uk/jdispatcher/msa/clustalo

Scoring Multiple Alignments

- Given sequences $A_1[0..n_1), \ldots, A_k[0..n_k)$ over common alphabet Σ
- alignment is sequence of *columns* in $(\Sigma_-)^k$ with $\Sigma_- = \Sigma \cup \{-\}$
- going from 2 to k sequences requires score for k-columns
  - different options
  - One option: total Hamming distance (see Unit 2 for motifs)
  - Here: ***SP-Score*** *(sum-of-pairs score)* w.r.t. S

$$d_{SP}\left(\begin{bmatrix} c_1 \\ \vdots \\ c_k \end{bmatrix}\right) = \sum_{1 \le i < j \le k} S\left(\begin{bmatrix} c_i \\ c_j \end{bmatrix}\right)$$

for S any pairwise-alignment score



Motif with consensus and profile

Motifs	T	C	G	G	G	G	g	T	T	T	t	t
	c	C	G	G	t	G	A	c	T	T	a	C
	a	C	G	G	G	G	A	T	T	T	t	C
	T	t	G	G	G	G	A	c	T	T	t	t
	a	a	G	G	G	G	A	c	T	T	C	C
	T	t	G	G	G	G	A	c	T	T	C	C
	T	C	G	G	G	G	A	T	T	c	a	t
	T	C	G	G	G	G	A	T	T	c	C	t
	T	a	G	G	G	G	A	a	c	T	a	C
	T	C	G	G	G	t	A	T	a	a	C	C

SCORE(*Motifs*) $3 + 4 + 0 + 0 + 1 + 1 + 1 + 5 + 2 + 3 + 6 + 4 = 30$

COUNT(*Motifs*)	A:	2	2	0	0	0	0	9	1	1	1	3	0
	C:	1	6	0	0	0	0	0	4	1	2	4	6
	G:	0	0	10	10	9	9	1	0	0	0	0	0
	T:	7	2	0	0	1	1	0	5	8	7	3	4
PROFILE(*Motifs*)	A:	.2	.2	0	0	0	0	.9	.1	.1	.1	.3	0
	C:	.1	.6	0	0	0	0	0	.4	.1	.2	.4	.6
	G:	0	0	1	1	.9	.9	.1	0	0	0	0	0
	T:	.7	.2	0	0	.1	.1	0	.5	.8	.7	.3	.4
CONSENSUS(*Motifs*)		T	C	G	G	G	G	A	T	T	T	C	C

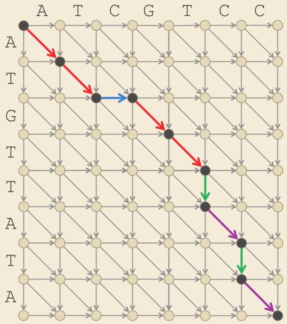
Compeau & Pevzner, *Bioinformatics Algorithms*, Fig. 2.2
https://cogniterra.org/lesson/29868/step/9?unit=21966

Dynamic Programming Solution

Pairwise alignment = path in grid graph;   optimal alignment = shortest path between corners

```
0   1   2   2   3   4   5   6   7   8
  A   T   -   G   T   T   A   T   A
  A   T   C   G   T   -   C   -   C
0   1   2   3   4   5   5   6   6   7
  ↘   ↘   →   ↘   ↘   ↓   ↘   ↓   ↘
```

match/**mismatch** (↘/↘), **insertion** (→), or **deletion** (↓).



Compeau & Pevzner, *Bioinformatics Algorithms*, Fig. 5.5 & 5.6
`https://cogniterra.org/lesson/29932/step/1?unit=22029`

$\rightsquigarrow$ DP solution with 2D matrix $D[0..m][0..n]$

For k strings, shortest path in k-dimensional grid graph

$\rightsquigarrow$ $n_1 \cdot n_2 \cdot \cdots \cdot n_k$ vertices to consider   for k strings of n characters $\Theta(n^k)$ time ↯

Bad News (Again)

MULTIPLE ALIGNMENT WITH SP-SCORE is NP-hard for any $\sigma \geq 2$ and any metric S

Elias: *Settling the Intractability of Multiple Alignment*, J. of Computational Biology **2006**

Proof Idea: Reduction from VERTEX COVER ON CUBIC GRAPHS

Bounding SP-scores

Not all hope is lost.

SP-score can be bounded by optimal pairwise alignments and heuristic for some alignment:

$$\sum_{1\le i<j\le k} d_S(A_i, A_j) \;\le\; d_{SP}(A_1,\ldots,A_k) \;\le\; d_{SP}(\text{some alignment})$$

- can be the basis for a Branch & Bound algorithm
- but: need efficient approximation algorithm for MULTIPLE ALIGNMENT WITH SP-SCORE

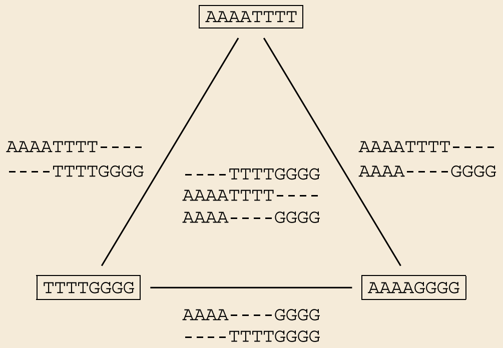
$\rightsquigarrow$ *Can we build a multiple alignment by successively adding in one new sequence at a time?*

Extending Pairwise Alignments is tricky

Can we combine optimal pairwise alignment into a multiple alignment?

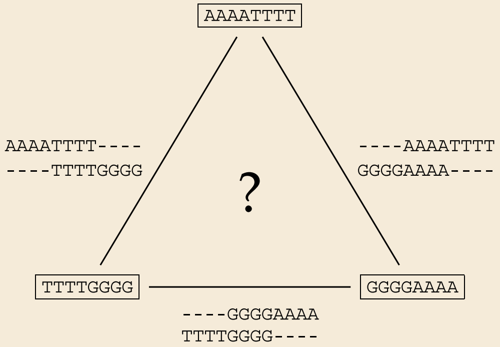
Sometimes Yes!

But No in general . . .



(a) Compatible pairwise alignments

Jones & Pevzner, *Bioinformatics Algorithms*, Fig 6.22a



(b) Incompatible pairwise alignments

Jones & Pevzner, *Bioinformatics Algorithms*, Fig 6.22b

Alignment Trees

Problem in example comes precisely from cycle!

- Given a *tree* over sequences $A_1, \dots, A_k$
- Compute optimal *pairwise* alignments along all $k-1$ tree edges
- Build multiple alignment one edge at a time
  - Here, use $\left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]$ for every gap symbol in either endpoint of an edge
    We always assume $S(\left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]) = 0$

- **Notation:**
  - $M \in (\Sigma_{-}{}^{k})^{N}$ multiple alignment of length $N \geq \max n_j$
  - d_{SP} SP-Score w.r.t. pairwise score S
  - $d_S(A, B)$ score of optimal pairwise alignment of A and B
  - M induces pairwise alignment $M[:][i, j]$ for A_i and A_j
    Note: $S(M[:][i, j]) \geq d_S(A_i, A_j)$ and in general not optimal

Center-Star Approximation

Use simplest possible tree: A *star*!

Center-Star Multiple Sequence Alignment

1. Compute all pairwise distances $d_S(A_i, A_j)$
2. Find $c \in [k]$ that minimizes $\sum_j d_S(A_c, A_j)$
3. Construct M as alignment consistent with star alignment with center S_c.

Center-Star Approximation – Analysis

Theorem 3.4
Assume d_S is a metric for pairwise alignments. The center-star alignment for k strings is a $(2 - \frac{2}{k})$-approximation w.r.t. to the SP-score of the multiple sequence alignment. ◀