Chapter 3-Pairwise Sequence Alignment - Unveiling Evolutionary Stories from Biological Sequences

Reza Rezazadegan
Shiraz University
www.dreamintelligent.com

Learning Outcomes:

- **Define** pairwise sequence alignment and its fundamental role in bioinformatics.
- **Differentiate** between sequence similarity, identity, and homology, and discuss the "twilight zone" of sequence alignments.
- **Compare and contrast** global and local alignment strategies, identifying appropriate applications for each.
- **Explain** the principles of the dot matrix method, and details of dynamic programming (Needleman-Wunsch and Smith-Waterman), for sequence alignment.
- **Describe** the construction and application of substitution matrices (PAM and BLOSUM) and various gap penalty schemes.
- **Assess** the statistical significance of alignment scores using P-values.
- **Discuss** practical challenges in sequence alignment, including computational efficiency, low-complexity regions, and the choice between DNA and protein sequence comparison.
- **Utilize** Biopython tools for performing and analyzing pairwise sequence alignments.

1. Introduction: The Language of Life and How We Read It

Bioinformatics, at its heart, is about making sense of the vast amount of biological data generated by modern sequencing technologies. One of the most fundamental operations in this field is **Pairwise Sequence Alignment (PSA)**, a technique used to compare two biological sequences – whether they are *DNA, RNA, or protein* – by searching for common character patterns and establishing a residue-to-residue correspondence. This process is not merely an academic exercise; it's the bedrock upon which more sophisticated analyses, such as *database similarity searching* and *multiple sequence alignment*, are built.

The ultimate goal of PSA is to **find the best possible pairing of two sequences** that maximizes the agreement among their residues. This often requires introducing *"gaps"* into one or both sequences to account for insertions or deletions that have occurred over evolutionary time.

Imagine trying to compare two ancient manuscripts that were copied from the same original, but *one copyist accidentally skipped a few words*, and another added some commentary. To find the true similarities, you'd need to shift parts of the text and insert blanks where words are missing. Sequence alignment does precisely this for biological sequences.

Note: *An* alignment between two sequences is any way of matching their bases, possibly with mismatches and inserting gaps. *Aligning* two sequences means finding the *best* possible alignment between the two, i.e. the one with the highest *alignment score*, to be defined precisely below.

Note: Biological sequences are a type of *sequential data*. Other types of such data include: text, code, musical notes and time series. As such, they can also be studied using the tools that are typically used for sequential data such as the *transformer neural network*. Student presentation topic

2. Evolutionary Basis and Key Concepts: Tracing Ancestry

The profound importance of PSA stems from its deep connection to **evolutionary theory**. Sequences that are significantly similar are not just coincidentally alike; they are generally presumed to share a **common evolutionary ancestor**, a relationship known as **homology**.

- Fundamental question in PSA: whether the similarities perceived between two sequences are due to chance, and are thus of little biological significance, or whether they are due to the derivation of the sequences from a common ancestral sequence, and are thus homologous.
- Because homology implies common ancestor, knowing the function/structure of one of two homologous two proteins, can help us infer the structure of the other.

- **Homology vs. Similarity – An Inferred Relationship**: It's crucial to understand the distinction between **similarity** and **homology**. **Similarity is a quantitative observation** derived directly from an alignment (e.g., "70% of residues match"). **Homology, however, is an inference about shared ancestry** based on that observed similarity. Two sequences are either homologous or not; there are no degrees of homology. Remember that evolution arises from the mutations that occur during DNA replications.
- Low sequence similarity does not necessarily rule out common function or homology.
- Sequences can also be significantly similar to each other, and yet not be evolutionarily homologous, as a result of *convergent evolution* for similar function.
- Comparisons of protein sequences show up homology more easily than comparisons of the corresponding DNA sequences: 4 nucleotides vs 20 amino acids, genetic code is redundant.
  - **Historical Context: The Hemoglobin Riddle**: In 1965, Emile Zuckerkandl and Linus Pauling published a groundbreaking paper, *proposing that DNA sequences could act as "Molecules as documents of evolutionary history."* Their discovery of high amino acid *sequence similarity between human and gorilla beta-hemoglobin* ignited debate. Many biologists, accustomed to anatomical comparisons, were *skeptical that mere molecular analysis could provide such profound evolutionary insights*. This "hemoglobin riddle" helped shift the scientific paradigm, establishing molecular data as a powerful tool for understanding evolutionary relationships. It's a prime example of how observed sequence similarity leads to the inference of homology.
- **Sequence Identity and Similarity – The Specifics**:
  - **Sequence identity** refers specifically to the percentage of *exactly matching* sequence residues in an alignment.
  - **Sequence similarity** is a broader measure, encompassing both identical matches and "conservative substitutions." *A conservative substitution occurs when one amino acid is replaced by another with similar physicochemical properties* (e.g., swapping isoleucine for valine, or phenylalanine for tyrosine, see Fig. 2.3 in Understanding Bioinformatics). This acknowledges that *not all mutations equally impact protein function*. For example, a small, nonpolar amino acid replacing another small, nonpolar amino acid is less likely to disrupt protein structure than a small nonpolar replacing a large charged one. See *Substitution Matrices* below.
- **The "Twilight Zone" and "Midnight Zone"**: The confidence with which we can infer homology decreases as sequence similarity drops.
  - For sequences *around 100 residues in length*, an identity of 30% or higher is generally considered to be in the "**safe zone**," providing *strong evidence for homology*.
  - Burkhard Rost found that 90% of sequence pairs with identity at or greater than 30% over their whole length were *pairs of structurally similar proteins*.
  - Identities between 20% and 30% fall into the "**twilight zone**," where it becomes challenging to differentiate true, distant homologs from sequences that appear *similar by mere chance*.
  - Below 20% identity is the "**midnight zone**," where homologous relationships cannot be reliably determined from sequence similarity alone. These zones highlight the statistical nature of inferring evolutionary relationships from sequence data.
- **Insertions and Deletions (Indels)**: Biological sequences are dynamic; over evolutionary time, they acquire **insertions and deletions (indels)** of genetic material. To create an alignment that reflects the most accurate evolutionary scenario and maximizes observable similarity, **gaps** are strategically introduced into one or both sequences. These **gaps** represent the *hypothesized positions where indels occurred* in one lineage relative to another. For example, in the alignment `A T - G T T A T A` versus `A T C G T - C - C` , the hyphens denote indels.

```
T H I S I S A - S E Q U E N C E
| |           |   |   |   |   |   |   |
T H - - - - A T S E Q U E N C E
```

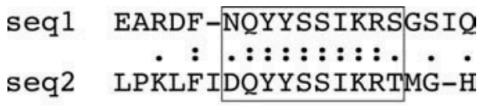
3. Alignment Strategies: Global vs. Local Search

The choice of alignment strategy is critical and depends on the presumed relationship between the two sequences being compared.

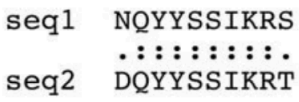
1. **Global Alignment**:
   - **Assumption**: This strategy is applied when it is assumed that the two sequences are **generally similar along their entire lengths**, *implying a shared evolutionary history throughout*.
   - **Process**: The algorithm attempts to find the best possible alignment that spans the *full length* of both sequences, from their very first residue to their very last. In terms of dynamic programming, the alignment path must extend from one corner of the

scoring matrix (usually top-left) to the opposite corner (bottom-right).

- **Application**: Global alignment is most suitable for **closely related sequences of approximately the same length**, such as *orthologous genes from closely related species*.
- **Limitation**: Its main drawback is that it can fail to identify highly similar *local* regions if the sequences are otherwise largely divergent or of significantly different lengths. By forcing an alignment across dissimilar regions, it may obscure important, limited similarities.
- **Example Program**: The **Needleman-Wunsch algorithm** is the classical dynamic programming approach for global alignment. A web-based program called **GAP** (Global Alignment Program) performs global pairwise alignment and often avoids penalizing terminal gaps, which can be useful when comparing sequences that truly differ in length but are homologous. Biopython's `PairwiseAligner` can perform global alignments by setting `mode = "global"`.



global sequence alignment



local sequence alignment

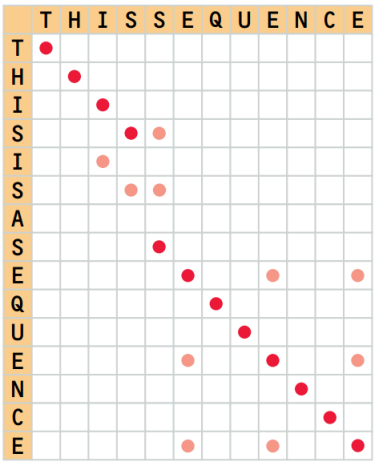
2. **Local Alignment**:
    - **Assumption**: This strategy is used *when sequences may not be globally related* but are expected to share **highly conserved local regions**, such as functional domains, motifs, or short gene fragments.
    - **Process**: The algorithm focuses exclusively on finding and aligning **only the segments with the highest levels of similarity**, effectively *ignoring the less similar or unrelated parts of the sequences*. In a dynamic programming matrix, *the alignment path can begin and end anywhere along the main diagonal*, focusing on local maxima.
    - **Application**: Local alignment is more appropriate for **divergent biological sequences**, sequences containing multiple domains of different origins, or sequences of disparate lengths. It is invaluable for *searching for conserved patterns* or *functional domains* within sequences.
    - **Example Programs**: The **Smith-Waterman algorithm** is the classical dynamic programming approach for local alignment. Biopython's `PairwiseAligner` can perform local alignments by setting `mode = "local"`.

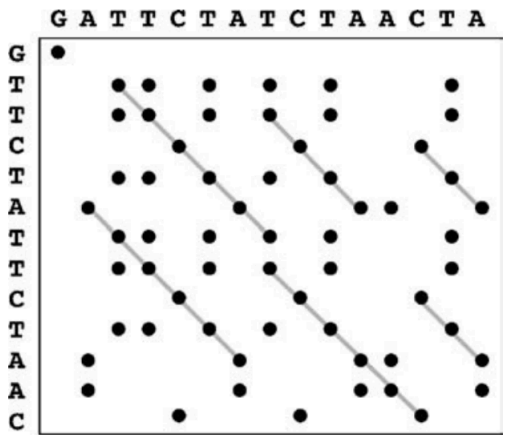
4. Algorithms for Pairwise Sequence Alignment

Three primary algorithmic approaches are used for PSA, each with unique strengths and computational characteristics:

1. **Dot Matrix Method**:
    - **Description**: This is a visual and intuitive technique that constructs a **two-dimensional matrix**. One sequence is placed along the horizontal axis, and the other along the vertical. A dot is placed at every intersection *where residues from both sequences match*. Does not use gaps.
    - **Interpretation**: Regions of similarity appear as **contiguous diagonal lines of dots**. Direct repeats within a single sequence appear as diagonals away from the main diagonal, while *inverted repeats or palindromic sequences* can appear as diagonals in a dot plot of a sequence against its reverse complement.
    - **Advantages**: Excellent for rapid visual identification of similar regions, repeats, and features like self-complementarity, which might indicate secondary structures.
    - **Limitations**: It typically requires *manual interpretation* to construct a full alignment with gaps, lacks statistical rigor, and is not easily scalable for multiple sequence alignment.

- **Example Program**: Dothelix is a web server that performs dot matrix alignments.





The dots in red, which form diagonal lines, represent runs of matched residues.

2. **Dynamic Programming (DP) Method**:
   - **Core Principle**: This is a **rigorous quantitative method** that guarantees finding optimal alignment(s) by systematically *considering all possible pairings of characters* between two sequences. It achieves this by breaking the large problem into smaller, overlapping subproblems, solving each once, and storing the intermediate solutions in a scoring matrix. The goal is to *find the optimal path through a grid, representing the "most appropriate correspondence of symbols*" between sequences.
3. **Word Method**:
   - This is a **heuristic approach** developed primarily for **accelerating database similarity searches**. This method is a key component of popular database search tools like **FASTA** and **BLAST**, discussed in the next chapter.

5. Dynamic Programming Method

The dynamic programming (DP) algorithm is the core technique used to find the optimal alignment between two biological sequences by maximizing their *similarity score*. This rigorous method efficiently explores all possible alignments, guaranteeing the identification of the optimal alignment for a given scoring scheme.

If A is an alignment between two sequences σ_1, σ_2 then at any index i it matches a base or a gap $A_1(i)$ in the first sequence with a base or a gap $A_2(i)$ in the second sequence. (Two gaps are not allowed to be matches in PSA.) The *score* of this alignment is defined as

$$S(A) = \sum_i Score(A_1(i), A_2(i))$$

where $Score(x, y)$ is given by:

- A *gap penalty* if one of the x, y is a gap.
- Otherwise it is given by a match score for nucleotides, and an entry in a *substitution matrix* for proteins.

These terms are defined further below.

The dynamic programming approach frames sequence alignment as an instance of the **Longest Path in a Directed Acyclic Graph (DAG) Problem**.

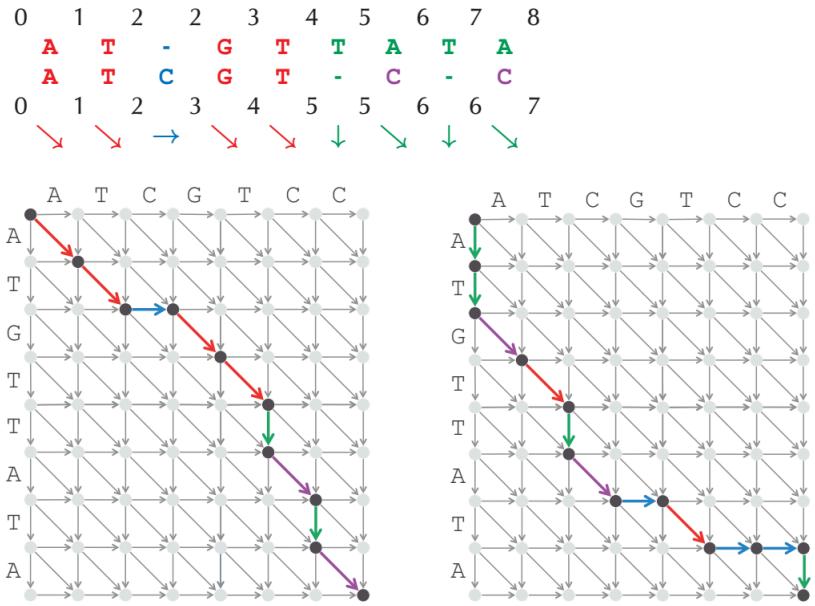


Image Credit: Compeau, Pevtzner, Bioinformatics Algorithms

5.1. Modeling Sequence Alignment as a Graph Problem

The alignment process begins by constructing an **alignment graph** (grid) for the two sequences being compared, v and w.

- **Nodes:** The graph is an $(|v|+1) \times (|w|+1)$ rectangular grid where nodes (i, j) represent the alignment state up to the i-prefix of sequence v and the j-prefix of sequence w.
- **Edges (Alignment Steps):** Paths within this grid represent possible alignments, with edges corresponding to three types of alignment operations:
  1. **Diagonal Edges:** Connect $(i-1, j-1)$ to (i, j) and represent aligning residue v_i with w_j, which is either a match or a mismatch.
  2. **Horizontal Edges:** Connect $(i, j-1)$ to (i, j) and represent an **insertion** (a gap in v).
  3. **Vertical Edges:** Connect $(i-1, j)$ to (i, j) and represent a **deletion** (a gap in w).
- **Weights and Scoring:** *Each edge is assigned a weight based on a defined **scoring matrix** (substitution matrix) and gap penalties*. The objective of the DP algorithm is to find the path from the source node (0, 0) to the sink node $(|v|, |w|)$ that yields the maximum *total score*.

5.2. The Dynamic Programming Recurrence

The DP algorithm computes the optimal alignment score iteratively by storing the optimal scores of subsequences in a matrix S. Let $S_{i,j}$ denote the score of the *highest scoring path* from the source (0, 0) to node (i, j).

The score $S_{i,j}$ for any node (i, j) is determined by the score of its predecessor nodes $(i-1, j)$, $(i, j-1)$, and $(i-1, j-1)$.

The fundamental recurrence relation for computing $S_{i,j}$ in a generalized sequence alignment (like the **Global Alignment Problem**) with linear gap penalties is defined as the maximum of the scores resulting from the three possible incoming paths:

$$S_{i,j} = \max \begin{cases} S_{i-1,j} + \text{Score}(v_i, -) & \text{(Deletion, using the vertical edge)} \\ S_{i,j-1} + \text{Score}(-, w_j) & \text{(Insertion, using the horizontal edge)} \\ S_{i-1,j-1} + \text{Score}(v_i, w_j) & \text{(Match/Mismatch, using the diagonal edge)} \end{cases}$$

Here $-$ represents a gap and the $Score()$ are the *scores or penalties for gaps, matches and mismatches*, discussed below.

The calculation proceeds from the starting point ($S_{0,0}$) outward or upward, solving the smaller problems once and storing the result to compute larger problems.

5.3. Backtracking and Alignment Reconstruction

Once the scoring matrix S (or matrices, in the case of affine gaps) is filled, the final alignment path is constructed via **backtracking** (or traceback).

1. To enable reconstruction, the algorithm typically stores **backtracking pointers** (e.g., in a matrix $Backtrack$) at each node (i, j) indicating *which of the three predecessors yielded the maximum score*. The pointers usually take one of three values
2. The traceback procedure starts at the final node (the sink $S_{|v|,|w|}$ for global alignment, or the highest-scoring cell for local alignment) and follows the pointers back to the source (0, 0). This reverse path determines the residues that constitute the optimal alignment.

5.4. Variations for Global and Local Alignments

The general DP approach is modified slightly to solve specific alignment tasks:

Global Alignment (Needleman-Wunsch)

The Needleman-Wunsch algorithm finds the optimal alignment over the entire length of the two sequences.

- **Initialization:** The scores $S_{i,0}$ (for the first row) and $S_{0,j}$ (for the first column) are initialized according to the accumulated gap penalties, as alignment at these positions requires aligning prefixes against only gaps.
- **Traceback:** The optimal path is guaranteed to span the entire matrix, starting at the source $(0, 0)$ and ending at the sink $S_{|v|,|w|}$.

Local Alignment (Smith-Waterman)

The Smith-Waterman algorithm is designed to identify the *highest-scoring conserved substrings* (local alignments) within two longer sequences.

- **Initialization and Score Restriction:** The computation of the DP matrix for local alignment is similar to the one for global alignment with the difference that in local alignment the score of any matrix entry that is negative, is set to zero. This is because a negative entry means that the alignment before that entry has not been well and setting it to zero allows it to be a *start*. In other words, in local alignment we have:

$$S_{i,j} = \max \begin{cases} 0 \\ S_{i-1,j} + Score(v_i, -) \\ S_{i,j-1} + Score(-, w_j) \\ S_{i-1,j-1} + \text{Score}(v_i, w_j) \end{cases}$$

- **Traceback:** Backtracking begins at the single *matrix element $S_{i,j}$ that has the overall maximum score in the entire matrix.* (This element represents the best-scoring end point of the optimal local alignment segment.) It continues backward until an element with a score of zero (a local "start") is reached.

5.6. Computational Efficiency

The runtime of the dynamic programming algorithm for aligning two strings of lengths n and m is proportional to the number of edges in the alignment graph, resulting in a complexity of $\mathbf{O(n \cdot m)}$. The memory required to store the backtracking pointers for reconstruction is also $\mathbf{O(n \cdot m)}$.

For very long sequences, such as whole genomes, the quadratic memory requirement can be prohibitive. Techniques exist to reduce this requirement:

- **Linear Space Alignment:** If the goal is only to compute the alignment score (*not the path*), the memory requirement can be reduced to $\mathbf{O(n)}$ because only the previous column's scores are needed to compute the current column's scores.
- **Divide-and-Conquer:** To recover the actual alignment in linear space ($\mathbf{O(n)}$ memory), a divide-and-conquer strategy is used (e.g., the Middle Node/Middle Edge Problem). This involves finding a node (or edge) on the optimal path that crosses the middle column of the matrix, recursively dividing the problem into two smaller alignment problems. While achieving linear space, this technique requires calculating scores in the forward and reverse directions, resulting in a runtime that is still $\mathbf{O(n \cdot m)}$.

6. Scoring the Mutations and Gaps

The DP algorithm discussed above, depends on the *gap penalties* $Score(-, v), Score(u, -)$, as well as the match or mismatch scores $Score(u, v)$. The gap penalties are discussed in the next subsection. For mutations, the scores are derived from statistical analysis of residue substitution data from sets of reliable alignments of highly related sequences.
For nucleotides, the score $Score(u, v)$ is decided simply based on whether we have a match $u = v$ (a positive score) or a mismatch $u \neq v$ (a negative score). This is assuming that the frequency of mutations is the same for all pairs of nucleotides.

For proteins however, changing one amino acid to another may not change the properties of the protein much, as some amino acids are similar to each other in terms of structure and chemical properties. Thus for proteins we need *substitution matrices* that encode the scores for the substitution of one amino acid with another.

1. **Substitution Matrices (for amino acids)**:
   - **Description**: These are tables that assign numerical scores to every possible pair of aligned *amino acid* residues, reflecting the **likelihood of one residue being substituted by another** over evolutionary time. They are derived from statistical analyses of observed substitutions in *trusted alignments of genuinely related sequences*. Scores are often expressed as **log-odds ratios**, which are *logarithmic ratios of the observed mutation frequency divided by the probability of substitution expected by random chance.*
   - A *positive score means that the frequency of amino acid substitutions found in a data set of homologous sequences is greater than would have occurred by random chance* (suggesting evolutionary conservation), so they represent substitutions of very similar or identical residues. A zero score means that the frequency of amino acid substitutions found in the homologous sequence data set is equal to that expected by chance.

- **PAM (Point Accepted Mutation) Matrices**:
  - **Historical Context**: The earliest and most influential substitution matrices were developed by **Margaret Dayhoff** and her colleagues in the 1960s and 1970s.
  - **Derivation**: The **PAM1 matrix** is based on observed *substitutions in very closely related proteins*, specifically those with only 1% amino acid divergence. The element $M(a, b)$ in the **Mutation Probability Matrix** gives the probability (or frequency) that residue type b will be replaced by (or mutated to) residue type a, among sequences with only 1% amino acid divergence. This matrix M is not symmetrical. Note that 1% amino acid divergence is regarded here as an *evolutionary unit*.
  - **Extrapolation**: Other PAM matrices (e.g., PAM250, PAM80) are **extrapolated** from PAM1 using an evolutionary model. *PAM250, for instance, theoretically models sequences that have undergone 250 point mutations per 100 residues*, representing greater evolutionary distance. Matrices for higher evolutionary distances (called PAM n) are obtained by **multiplying the $\mathbf{M}$ matrix by itself $\mathbf{n}$ times ($\mathbf{M^n}$)**. For example, PAM2 represents 2 mutations per 100 residues and is derived by raising the 1-PAM matrix to the 250th power.

ORIGINAL AMINO ACID

REPLACEMENT AMINO ACID	A Ala	R Arg	N Asn	D Asp	C Cys	Q Gln	E Glu	G Gly	H His	I Ile	L Leu	K Lys	M Met	F Phe	P Pro	S Ser	T Thr	W Trp	Y Tyr	V Val
A Ala	9867	2	9	10	3	8	17	21	2	6	4	2	6	2	22	35	32	0	2	18
R Arg	1	9913	1	0	1	10	0	0	10	3	1	19	4	1	4	6	1	8	0	1
N Asn	4	1	9822	36	0	4	6	6	21	3	1	13	0	1	2	20	9	1	4	1
D Asp	6	0	42	9859	0	6	53	6	4	1	0	3	0	0	1	5	3	0	0	1
C Cys	1	1	0	0	9973	0	0	0	1	1	0	0	0	0	1	5	1	0	3	2
Q Gln	3	9	4	5	0	9876	27	1	23	1	3	6	4	0	6	2	2	0	0	1
E Glu	10	0	7	56	0	35	9865	4	2	3	1	4	1	0	3	4	2	0	1	2
G Gly	21	1	12	11	1	3	7	9935	1	0	1	2	1	1	3	21	3	0	0	5
H His	1	8	18	3	1	20	1	0	9912	0	1	1	0	2	3	1	1	1	4	1
I Ile	2	2	3	1	2	1	2	0	0	9872	9	2	12	7	0	1	7	0	1	33
L Leu	3	1	3	0	0	6	1	1	4	22	9947	2	45	13	3	1	3	4	2	15
K Lys	2	37	25	6	0	12	7	2	2	4	1	9926	20	0	3	8	11	0	1	1
M Met	1	1	0	0	0	2	0	0	0	5	8	4	9874	1	0	1	2	0	0	4
F Phe	1	1	1	0	0	0	0	1	2	8	6	0	4	9946	0	2	1	3	28	0
P Pro	13	5	2	1	1	8	3	2	5	1	2	2	1	1	9926	12	4	0	0	2
S Ser	28	11	34	7	11	4	6	16	2	2	1	7	4	3	17	9840	38	5	2	2
T Thr	22	2	13	4	1	3	2	2	1	11	2	8	6	1	5	32	9871	0	2	9
W Trp	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	9976	1	0
Y Tyr	1	0	3	0	3	0	1	0	4	1	1	0	0	21	0	1	1	2	9945	1
V Val	13	2	1	1	3	2	2	3	3	57	11	1	17	1	3	2	10	0	2	9901

Figure 82. Mutation probability matrix for the evolutionary distance of 1 PAM. An element of this matrix, M_{ij}, gives the probability that the amino acid in column j will be replaced by the amino acid in row i after a given evolutionary interval, in this case 1 accepted point mutation per 100 amino acids. Thus, there is a 0.56% probability that Asp will be replaced by Glu. To simplify the appearance, the elements are shown multiplied by 10,000.

PAM1 matrix from the original paper

- **Scoring Matrix Calculation:** The final PAM scoring matrix is derived using the **log-odds ratio**. The entry $\mathrm{PAM}_n(i,j)$ is calculated using the formula where Here $f(j)$ is the background probability of amino acid j in the database. The logarithmic transformation is often taken to the **base 10**.

$$\log\left(\frac{M^n(i,j)}{f(j)}\right)$$

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	12																			
S	0	2																		
T	-2	1	3																	
P	-3	1	0	6																
A	-2	1	1	1	2															
G	-3	1	0	-1	1	5														
N	-4	1	0	-1	0	0	2													
D	-5	0	0	-1	0	1	2	4												
E	-5	0	0	-1	0	0	1	3	4											
Q	-5	-1	-1	0	0	-1	1	2	2	4										
H	-3	-1	-1	0	-1	-2	2	1	1	3	6									
R	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6								
K	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5							
M	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6						
I	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5					
L	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-2	4	2	6				
V	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4			
F	-4	-3	-3	-5	-4	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9		
Y	0	-3	-3	-5	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	7	10	
W	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17

The PAM250 matrix. The numbers are rounded to the nearest integer.

- **Application**: Higher PAM numbers (e.g., PAM250) are chosen for aligning **more divergent sequences**, while lower PAM numbers (e.g., PAM1) are for **closely related sequences**. They are often used for reconstructing phylogenetic trees due to their explicit

evolutionary model.

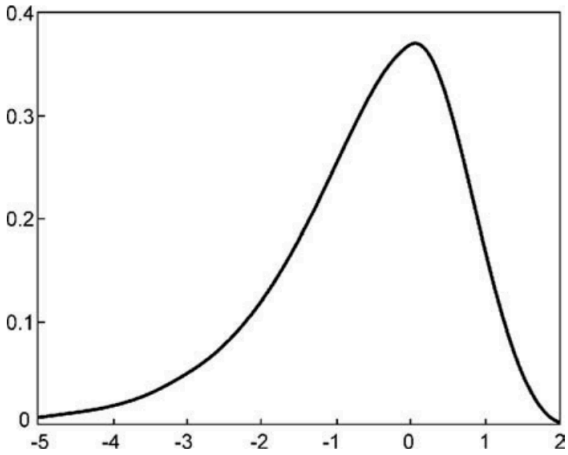
- **BLOSUM (Blocks Substitution Matrix) Matrices**:
  - **Derivation**: Developed by **Henikoff and Henikoff** , these matrices are based on the **direct observation** of amino acid *substitutions within conserved, ungapped blocks of multiple sequence alignments.* These "blocks" are highly conserved regions, typically less than 60 residues, where sequence alignments are highly reliable.
    - Sequences within these blocks are clustered based on a percentage identity threshold e.g., a BLOSUM62 matrix is derived from *blocks where sequences share 62% or more identity*. Substitutions are calculated *between* these clusters, rather than within them, to reduce bias from highly similar sequences.
  - **Application**: Conversely to PAM, **lower BLOSUM numbers** (e.g., BLOSUM45) are used for **more divergent sequences**, and higher numbers (e.g., BLOSUM90) for **closely related sequences**. *BLOSUM matrices are particularly advantageous for database searches and for identifying conserved protein domains.*
- **Comparison of PAM and BLOSUM**: A key difference is that most PAM matrices are derived by *extrapolation* from an evolutionary model, whereas BLOSUM matrices are based on *direct observation* from conserved blocks. This makes *PAM more suitable for tracing long evolutionary histories* and BLOSUM better for *identifying conserved regions* in local alignments and database searches. High PAM numbers (e.g., PAM250) for divergent sequences, but low BLOSUM numbers (e.g., BLOSUM45) for divergent sequences.
- **Biopython Implementation**: Biopython's `PairwiseAligner` allows users to specify custom `match_score` and `mismatch_score` (e.g., +5 for match, -4 for mismatch) or to load pre-defined substitution matrices like BLOSUM62 using `Bio.Align.substitution_matrices.load("BLOSUM62")` . It also supports "generalized alignments" for comparing lists of arbitrary objects, such as three-nucleotide codons.

2. **Gap Penalties**:
   - **Biological Justification**: Gaps are introduced in alignments to account for evolutionary *indel* events. Penalizing gaps ensures that *alignments with fewer, larger gaps are generally preferred over those with many small, dispersed gaps*. This reflects that a single indel event is often more probable than multiple independent indel events.
   - **Types of Penalties**:
     - **Constant Gap Penalty**: Assigns the same penalty score regardless of the gap's length. This is less biologically realistic.
     - **Linear Gap Penalty**: The penalty is directly proportional to the length of the gap.
     - **Affine Gap Penalty**: This is generally the most realistic model. It differentiates between a **gap opening penalty (GOP)** (a larger penalty for *starting* a new gap) and a **gap extension penalty (GEP)** (a smaller penalty for *extending* an existing gap). The total penalty for a gap of length *n* is often calculated as `open_gap_score + (n-1) * extend_gap_score` .
   - **Terminal Gaps**: Gaps occurring at the very beginning or end of an alignment are often treated with *no penalty*, especially when aligning sequences of different lengths. This is because they may not represent true evolutionary indel events but *simply differences in sequence boundaries*. Biopython's `PairwiseAligner` offers granular control over these, allowing distinct penalties for internal, left, and right gaps (e.g., `target_internal_open_gap_score` , `query_left_extend_gap_score`).

6. Statistical Significance of Sequence Alignment: Beyond Chance

After an alignment score is obtained, it is crucial to determine if that score is **statistically significant** – meaning, is it genuinely indicative of homology, or could it have arisen merely by random chance? This is not always straightforward, as *random matches can occur*, especially in long sequences or large databases. Note: even two randomly generated sequences may align at some bases.

- **Score Distribution**: The *distribution of similarity scores obtained from aligning unrelated sequences* typically follows the **Gumbel extreme value distribution**. (An *extreme value distribution* is the distribution of the maximum or minimum values of a quantity.)



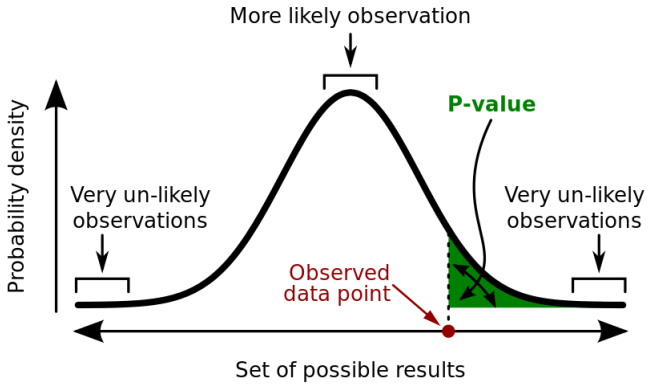
The Gumble distribution. The x-axis units are standard deviations.

Cumulative Distribution Function (CDF) of Gumbel: The probability $P(S < x)$ of the alignment score S being less than a value x, according to Gumbel Distribution, is given by

$$P(S < x) = \exp\left(-kmn \cdot e^{-\lambda x}\right)$$

where m, n are sequence lengths and λ, k are constants which have to be fitted. (They depend on substitution matrix and gap penalties used.) To estimate these constants, the distribution of the alignment scores of random shuffles of the sequences is considered.

- **P-value**: The **P-value** indicates the *probability that an alignment score better than or equal the observed score would occur solely due to random chance.* A low P-value (e.g., < 0.05) suggests that the alignment is statistically significant.



The probability of obtaining an alignment of score S greater than or equal to x is approximately

$$P(S \geq x) = 1 - \exp\left(-Kmne^{-\lambda x}\right) \simeq Kmne^{-\lambda x}$$

- If a P-value is smaller than 10e−100, it indicates an exact match between the two sequences.
- If the P-value is in the range of 10e−50 to 10e−100, it is considered to be a nearly identical match.
- A P-value in the range of 10e−5 to 10e−50 is interpreted as sequences having clear homology.
- A P-value in the range of 10e−1 to 10e−5 indicates possible distant homologs.
- If P is larger than 10e−1, the two sequence may be randomly related.

Sometimes truly related protein sequences may lack the statistical significance at the sequence level owing to fast divergence rates. Their evolutionary relationships can nonetheless be revealed at the three-dimensional structural level.

7. Computational Challenges and Practical Considerations

While dynamic programming algorithms guarantee optimal alignments, their computational demands (time and memory) can become **prohibitive** for very long sequences (e.g., entire genomes) or when comparing a query against a large database. This has led to the development of heuristic and approximation methods.

- **Computational Efficiency - The Need for Speed**:
  - **Memory Optimization**: For *extremely long sequences*, storing the entire dynamic programming matrix is impractical. Algorithms can be optimized to store only a few rows (e.g., two) of the matrix at a time, drastically reducing memory usage, though this usually makes the traceback procedure more complex and slower. This "algorithmic trick" is crucial for aligning complete bacterial genomes with limited resources.
  - **Reduced Matrix Calculation**: Techniques like **banded alignment** limit computations to a narrow band around the main diagonal of the scoring matrix, *assuming that optimal alignments generally do not stray far from this diagonal.*
  - **Rigorous vs. Heuristic**: While heuristic methods (like BLAST and FASTA) are fast, they do not guarantee optimal alignments.
- **DNA vs. Protein Alignment – The Power of Proteins**:
  - It is generally more **sensitive and informative** to align sequences at the **protein level** rather than the DNA level for inferring homology and conducting functional or evolutionary analyses.
  - **Alphabet Size and Information Content**: The 20-amino-acid alphabet of proteins provides more information per position than the 4-base alphabet of DNA, *making random matches less likely* and true biological signals more discernible. A match between two DNA sequences has a higher chance of occurring randomly (up to 50% identity for unrelated sequences with gaps) compared to protein sequences (only ~10%).
  - **Conservative Substitutions**: Protein substitution matrices (PAM, BLOSUM) are specifically designed to account for conservative amino acid changes, *capturing more nuanced evolutionary and physicochemical information than simple DNA scoring schemes*.
  - **Frameshifts**: Direct DNA alignment of protein-coding regions can introduce biologically unrealistic **"frameshift" errors** *if gaps are inserted without regard for codon boundaries*. (e.g. AU-GACC and AGCUUA) The recommended approach is often to **translate DNA sequences into their corresponding protein sequences, perform the alignment at the protein level, and then, if necessary, reverse-translate the alignment back to DNA** to preserve codon consistency.
- **Parameter Choice – Customizing the Search**: The outcome of any alignment is *highly dependent on the chosen substitution matrix, gap opening penalties, gap extension penalties*, and word size (for heuristic methods). It is advisable to *explore various parameter combinations, especially when dealing with sequences of unknown evolutionary distance*, to find the set that yields the most biologically meaningful alignment.
- **Low-Complexity Regions (LCRs) – Avoiding Spurious Hits**: These are sequence segments with a highly biased composition (e.g., *stretches of a single amino acid, simple repeats* like "RRRRRR"). LCRs can cause spurious, high-scoring alignments between *unrelated* sequences, thus obscuring true biological relationships. Programs often **mask** these regions (replacing them with 'X's for proteins or 'N's for DNA) to prevent misleading matches and improve the sensitivity of searches for genuine homologs.

8. Biopython for Pairwise Alignment: A Practical Toolkit

The **Biopython library** offers powerful and flexible tools for pairwise alignment.

- **The `Bio.Align.PairwiseAligner` Class**: This class is the primary tool for pairwise alignment in Biopython (superseding the deprecated `Bio.pairwise2` module).
  - **Algorithm Selection**: It implements global (Needleman-Wunsch), local (Smith-Waterman), and other algorithms like Gotoh (three-state) and Waterman-Smith-Beyer.
  - **Parameter Control**: Users have fine-grained control over alignment parameters :
    - `match_score` and `mismatch_score` can be set directly.
    - Pre-defined substitution matrices like BLOSUM62 can be easily loaded using `substitution_matrices.load("BLOSUM62")`.
    - Affine gap penalties are fully customizable, with distinct `open_gap_score` and `extend_gap_score` for the query and target sequences, and separate penalties for internal, left, and right terminal gaps (e.g., `target_internal_open_gap_score`, `query_left_extend_gap_score`). Users can also define custom gap scoring functions.
  - **Alignment Output**:
    - `aligner.score(target, query)` returns the best alignment score.
    - `aligner.align(target, query)` returns an iterator of `Alignment` objects, allowing access to one or all optimal alignments if multiple exist.
    - `Alignment` objects have properties like `len(alignment)` (always 2 for pairwise) and `alignment.shape` (length of alignment and number of columns).

    - Alignments can be printed in a human-readable format.
- **Generalized Pairwise Alignments**: `PairwiseAligner` supports aligning not just single-letter strings but also lists or tuples of arbitrary objects. This is useful for advanced applications, such as aligning three-nucleotide codons directly. A specific substitution matrix (e.g., SCHNEIDER) can be loaded for codon alignments, with an alphabet of three-letter codons.
- **Substitution Matrix Derivation**: The `alignment.substitutions` property of an `Alignment` object can report substitution frequencies between residues, which can then be used to calculate a custom substitution matrix. This is similar to how BLOSUM matrices were originally derived.

Conclusion:
Pairwise sequence alignment stands as a critical and continually evolving tool in bioinformatics. From the pioneering efforts of Dayhoff and Levenshtein to the advanced algorithms and software like Biopython's `PairwiseAligner`, this technique allows biologists to unravel the evolutionary stories encoded within biological sequences. By understanding its evolutionary underpinnings, algorithmic nuances, scoring complexities, and statistical interpretations, researchers can extract profound biological insights into sequence function, structure, and ancestry. The careful application of PSA, alongside other bioinformatics methods, continues to drive discoveries in genomics, proteomics, and our understanding of life itself.

Exercises

1. Design a Python data structure (class) for storing *an* alignment of two sequences of the same type.
2. Consider the following alignment of two protein sequences. Compute its alignment score by hand using the PAM250 substitution matrix, if the penalty of a gap is -5.
   S N I Y G
   G - Q A -
3. Add a member function to your alignment class from exercise one that computes the score of an alignment, using affine gap penalty. If should give the choice of two substitution matrices PAM250 and BLOSUM62 for protein sequences and allow the user to set the two parameters for affine gap penalty.
4. Write a python function that takes two sequences and produces all their possible alignments.
5. Align the two sequences ATTC and TTG *by hand* using the DP method. Set the scores for match, mismatch and a gap to be 1, -1, -0.5 respectively. Perform both local and global alignment.
6. Implement the DP algorithm for PSA.
7. Imagine the probability distribution for a quantity Q is given by the density function $2e^{-2Q}$ for $Q > 0$. What is the p-value for observing a sample with $Q = 4$ totally by chance?