Chapter 4 - Database Similarity Search for Sequences- Hunting for Homologs in the Genomic World

Reza Rezazadegan
Shiraz University
www.dreamintelligent.com

Learning Outcomes: Upon completing this chapter, students should be able to:

- **Define** database similarity searching and its critical role in inferring biological function and evolutionary relationships.
- **Articulate** the unique requirements (sensitivity, specificity, speed) of database searching and the trade-offs involved.
- **Explain** the principles and operational steps of major heuristic search algorithms (BLAST and FASTA), including their variants.
- **Compare and contrast** BLAST and FASTA, highlighting their algorithmic differences and respective strengths.
- **Interpret** statistical measures like E-values and bit scores to assess the significance of database hits.
- **Discuss** practical challenges such as low-complexity regions and the choice between DNA and protein sequence queries, along with strategies for addressing them.
- **Utilize** Biopython tools for programmatic execution and parsing of database search results.

1. Introduction: The Need for Speed and Insight in the Genomic Era

With the explosion of biological sequence data, particularly from high-throughput genome sequencing projects, assigning function and understanding the evolutionary context of newly discovered sequences has become a paramount challenge. **Database similarity searching** is a fundamental bioinformatics application that addresses this challenge: it is the process of rapidly and efficiently *comparing a query biological sequence (DNA, RNA, or protein) against the vast collection of sequences stored in biological databases.*

- **Purpose of Database Searching**: The primary purpose of this process is to
  - **Assign putative functions** to novel sequences
  - **Infer evolutionary relationships (homology)**
  - Gain structural insights.
    Finding similarities in sequence can often provide the *first clues to the type of protein a new gene encodes* and its possible function.
- **Historical context:** In 1983 a newly sequenced gene PDGF whose function is to stimulate cell growth was searched against all other genes known at the time. It was discovered that PDGF was very similar to the *v-sis gene* which is a virus gene that can cause a cancer-like reaction in infected human cells. After this discovery, searching all new sequences against sequence databases became the first order of business in genomics.
- **Unique Requirements for Database Searching**: Unlike simple pairwise alignment of two known sequences, database searching operates on a massive scale, necessitating specific performance criteria. Effective database search algorithms must balance:
  1. **Sensitivity**: The ability to identify as many true homologous sequences (*true positives*) as possible, including distantly related ones, avoiding "false negatives". (Similar to *recall* in machine learning.)
  2. **Selectivity (Specificity)**: The ability to accurately *distinguish true homologous sequences from unrelated ones* (false positives) that may show fortuitous similarities. (Similar to *precision* in machine learning.)
  3. **Speed**: The efficiency of the algorithm in returning results within a practical timeframe, given that databases contain millions of entries and grow daily.

2. Historical Context: From Manual Comparison to Algorithmic Shortcuts

Early sequence comparisons were often painstaking manual processes. The development of general pairwise alignment algorithms like Needleman-Wunsch (1970) and Smith-Waterman (1981) was a critical first step. However, *applying these rigorous, exhaustive algorithms to entire databases proved too slow.* An estimate from 1990s suggested that searching a 300,000-sequence database with a 100-residue query could take 2–3 hours with standard computer systems of the time. This bottleneck necessitated the development of **heuristic methods**.

- **Heuristic Methods**: These algorithms take "shortcuts" by reducing the search space, *examining only a fraction of all possible alignments*. While they do not guarantee finding the absolute optimal alignment or all true homologs, they provide results significantly faster (50–100 times faster than dynamic programming), with only a moderate, generally acceptable, compromise in sensitivity and specificity. This trade-off makes them indispensable for routine database searching by molecular biologists.

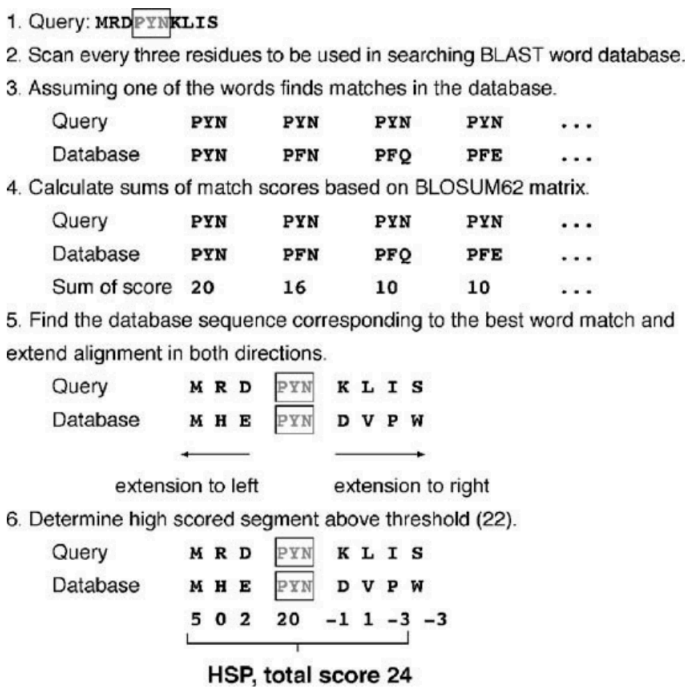
3. Heuristic Database Searching Algorithms: BLAST and FASTA

The two most widely used heuristic algorithms for database similarity searching are **BLAST** and **FASTA**. They both identify similar sequence segments to indicate sequence similarity.

3.1. BLAST (Basic Local Alignment Search Tool)

BLAST is arguably the most popular sequence database search program, developed by Stephen Altschul and colleagues at NCBI in 1990. Its paper is one of the most cited scientific papers ever published.

- **Core Principle**: BLAST's objective is to find **High-Scoring Segment Pairs (HSPs)**—*short, ungapped regions of high similarity between the query and database sequences*. The existence of such segments above a given threshold indicates pairwise similarity beyond random chance.
- **Procedure:**



1. **Word List Creation (Seeding)**: A list of all possible short "words" (k-mers) is generated from the query sequence. *For proteins, words are typically 3 residues long; for DNA, 11 residues*. For example for the query sequence MRDPYNKLIS, the 3-mers are MRD, RDP, DPY, PYN, YNK, NKL, KLI, LIS. There are at most 20^k k-mers for protein sequences. Crucially, *BLAST words need not be identical to query k-mers*, but must score above a given threshold T when aligned with a query k-mer using a substitution matrix. Thus, all the k-mers in the query sequence and the ones that are similar to those, above the given threshold, are gathered.
2. **Database Scanning**: The database is searched for occurrences of *all* these words. **Note:** a database is *indexed* for all types of 3-mers in advance, meaning that we have a *hash table* that maps each 3-mer, to the database sequences that contain it.
3. **Ungapped Extension**: Once matching words are found, the alignment is extended in both directions *without gaps*, accumulating a score using a substitution matrix (e.g., BLOSUM62). This extension stops when the score drops below a certain threshold due to mismatches.
4. **Gapped Extension (Newer Versions)**: The original BLAST algorithm did not consider gaps. However, *current versions allow gaps after an initial ungapped search*. In gapped BLAST, the highest scored segment is chosen to be *extended in both directions using dynamic programming where gaps may be introduced*. The extension continues if the alignment score is above a certain threshold; otherwise it is terminated. However, the overall score is allowed to drop below the threshold only if it is temporary and rises again to attain above threshold values.

- **Discussion of sensitivity and selectivity of BLAST**
- **BLAST Variants**: BLAST is a family of programs tailored for different sequence types :

- **BLASTN**: Nucleotide query vs. nucleotide database.
- **BLASTP**: Protein query vs. protein database.
- **BLASTX**: Nucleotide query (translated in six frames) vs. protein database.
- **TBLASTN**: Protein query vs. nucleotide database (translated in six frames).
- **TBLASTX**: Nucleotide query (translated in six frames) vs. nucleotide database (translated in six frames). This and TBLASTN are computationally intensive, because *all the 6 reading frames* are considered for each sequence. (BLAST doesn't know which frame is biologically relevant. So it checks all possible ones to catch homology.)
- **bl2seq**: Performs local alignment of two user-provided sequences.

- **Biopython for BLAST**: Biopython offers extensive tools for dealing with the volume of data generated by BLAST and automating runs. The `Bio.Blast.NCBIXML` parser is used to read XML output from BLAST searches, including RPS-BLAST. A `Blast Record` object stores information from a BLAST report, including program name, query ID, target database, and a quick overview of hits, including the number of HSPs.

3.2. FASTA (FAST ALL)

FASTA (FAST ALL) was the first database similarity search tool developed, preceding BLAST, by William Pearson and David Lipman in 1988.

- **Core Principle**: FASTA uses a "hashing" strategy to find matches for *short stretches of identical contiguous residues*, called "k-tuples" or "ktups". For proteins, k is typically 1 or 2; for DNA, 6. This is very *similar to a Dot Matrix*.
- **Procedure:**

1. Given two amino acid sequences for comparision:

```
sequence 1   AMPSDGL
sequence 2   GPSDNAT
```

2. Construct a hashing table:

amino acid	sequence position		offset
	seq 1	seq 2	
A	1	6	-5
D	5	4	1
G	6	1	5
L	7	-	-
M	2	-	-
N	-	5	-
P	3	2	1
S	4	3	1
T	-	7	-

3. Identify residues with the same offset values (highlighted in grey).

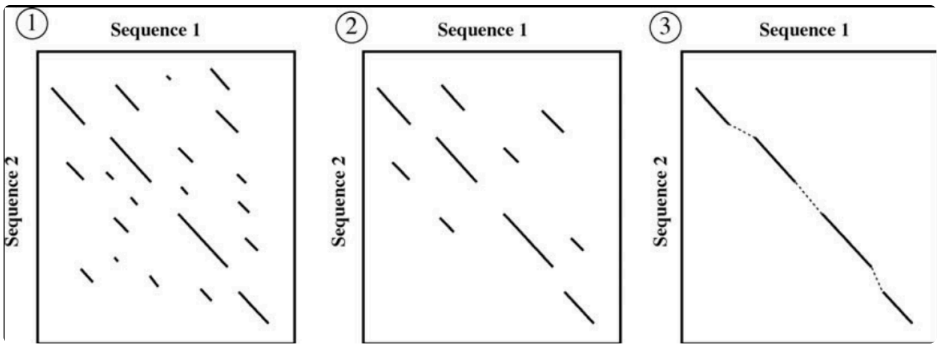
4. Find the matching word of three residues in the order of 3, 4 and 5 in one sequence and 2, 3,and 4 in the other.

5. This allows establishment of alignment between the two sequences.

```
sequence 1   AMPSDGL-
                |||
sequence 2   -GPSDNAT
```

1. **k-tuple Identification (Hashing)**: A lookup table is constructed for all k-tuples in the query sequence and their positions. This allows FASTA to quickly identify identical k-tuples in database sequences and their "offset" (positional difference). Ktups with the same offset form contiguous identical sequence regions that appear as stretches of diagonals in an alignment matrix.
2. **Rescoring and Ungapped Extension**: The top ten regions with the highest density of diagonals are identified. These are then *rescored using a substitution matrix* and extended without gaps. Only regions scoring above a certain threshold are retained.
3. **Joining High-Scoring Regions (Chaining)**: The program then attempts to *join neighboring high-scoring segments* along the same diagonal, allowing for gaps. This step involves applying gap penalties and calculating an initial gapped alignment score.
4. **Dynamic Programming Refinement**: The *highest-scoring database sequences from the previous steps are subjected to a final alignment using dynamic programming* (often a banded version of Smith-Waterman, which restricts the search to a narrow band around the diagonals) to produce the final, optimized alignment with gaps.

- **FASTA Variants**: Similar to BLAST, FASTA offers variants like FASTX (translates DNA query to protein for protein database search) and TFASTX (protein query vs. translated DNA database).



3.3. Comparison of FASTA and BLAST

Both BLAST and FASTA are highly effective for database searching, but they have key differences :

- **Seeding Mechanism**: BLAST uses a substitution matrix to find matching words, *allowing non-identical but similar k-mers*. FASTA, by default, identifies *identical* matching k-tuples using its hashing procedure.
- **Sensitivity and Speed**: *FASTA typically scans smaller window sizes* (ktups) by default, potentially making it *more sensitive* for certain comparisons but generally slower than BLAST.
- **Specificity**: BLAST's scoring threshold and rapid filtering remove many low-scoring noise hits early. It also extends only high-confidence seeds so it gets fewer false positives. Thus, *BLAST is generally more specific (or selective) than FASTA*.

4. Scoring and Statistical Significance: Distinguishing Real from Random

To infer homology from a database search, it is absolutely critical to assess the **statistical significance** of an alignment score – that is, whether it is genuinely indicative of biological relatedness or could have arisen purely by random chance. The test of significance here is similar the one one for pairwise sequence alignment with the difference that the larger the database, the more unrelated sequence alignments there are.

- **E-value (Expectation Value)**: represents the **number of alignments with a score as good as or better than the observed score that are expected to occur by chance** *in a database search of a given size*. It equals the sum of the p-values of aligning the query sequence with all the sequences in the database. If M is the total number of bases in the database, n the length of the query sequence and S, P the alignment score and p-value for the HSP alignment then:

  $$\simeq KMne^{-\lambda S}$$

  - **Interpretation**: A lower E-value indicates higher statistical significance. For example, *an E-value of 3.0 suggests that one would expect to find three such alignments by chance alone*, indicating the hit is likely unrelated. Conversely, very small E-values (e.g., 10p-20 or less) clearly indicate significant similarity. Default E-value thresholds are often set at 0.01 or 0.001.
  - **Limitation**: The E-value is proportionally affected by database size. As a database grows, the E-value for a given sequence match increases, potentially "losing" previously detected homologs.
- **Bit Score (S')**: is a **normalized pairwise alignment score that is independent of query sequence length and database size**. The bit-score is the *required size of a sequence database in which the current match could be found just by chance*. We have

  $$= Mn2^{-S}.$$

  It is derived from the raw alignment score and normalized using Gumbel distribution constants (λ and K).

  $$S = (\lambda S - \ln k)/\ln 2$$

  - **Interpretation**: A higher bit score indicates a more significant match. Because it is a constant (in terms of database size) statistical indicator, it is valuable for comparing search results across different databases or over time as databases expand.
- **Example: T. Rex and Chicken**: The process of determining statistical significance is beautifully illustrated by the "T. rex was just a big chicken?" story (Volume II of Bioinformatics Algorithms). When scientists found short peptide sequences (6-8 amino acids long) from a 68-million-year-old *T. rex* fossil, they realized that they were similar to peptides from bird collagen. Since two unrelated short sequences are likely to align by chance, they faced the question: could these matches have occurred by chance? By comparing the observed peptide matches against a large protein database (UniProt+) and calculating probabilities, they determined the expected number of peptides that would score as highly by chance. This real-world example underscores the necessity of statistical validation in bioinformatics. Student presentation topic

5. Practical Issues

Effective database searching goes beyond simply running BLAST or FASTA; it involves careful consideration of sequence characteristics, appropriate parameter selection, and leveraging advanced techniques.

- **Low-Complexity Regions (LCRs)**:
  - **Problem**: Many protein and DNA sequences contain regions of **low compositional complexity**—stretches of repetitive residues (1 or 2 residues) (e.g., poly-A runs, simple repeats). These LCRs are prevalent (estimated 15% of protein sequences in databases) and can cause spurious, high-scoring alignments between *unrelated* sequences, obscuring true biological relationships and violating the statistical assumptions of search algorithms.
  - For example if two sequence contain "AAAAAAA", this causes a high alignment score but is biologically *meaningless because this region has no complexity.*
  - Also the high alignment score from matching LCRs in two sequences can overshadow the nearby, more meaningful regions.
  - **Masking**: Programs like **SEG** (for proteins) and **DUST** (for DNA) detect and "mask" LCRs (replacing them with 'X's for proteins or 'N's for DNA) before database searches, preventing them from generating misleading matches and improving search sensitivity.
- **Parameter Choice**: The results of a search are highly dependent on the choice of substitution matrix (e.g., BLOSUM62 for general purpose, lower BLOSUM or higher PAM for divergent sequences), gap penalties, and word size. *These parameters should be chosen carefully based on the expected evolutionary distance and biological context.*
- **Database Selection**: The choice of which database to search depends on the specific research question. Generic protein or nucleic acid sequence databases (e.g., GenBank, SWISS-PROT) are good starting points. Specialized databases (e.g., dbEST for expressed sequences, PDB for structures, genome-specific databases, PROSITE for motifs) can provide more focused information.

6. Rigorous and Advanced Database Searching Methods

While heuristic methods are fast, they are not guaranteed to find optimal alignments and *can miss truly significant hits, especially for distantly related sequences* (BLAST can miss 30% of true hits for some families).

- **Rigorous Database Searching**: Advances in computational technology, particularly *parallel processing,* have made rigorous methods like the **Smith-Waterman algorithm** more feasible for database searching, offering maximum sensitivity.
  - **ParAlign** is such an algorithm and can align multiple residues at once.
- **Intermediate Sequences**: A strategy to find very distantly related homologs involves using "intermediate sequences". If query A matches protein X, and query B also matches protein X, then A and B might be homologous through their shared link to X, even if A and B don't directly align.

7. Biopython for Database Searching: Automating Your Workflow

Biopython provides robust tools for performing and parsing database similarity search results, addressing the challenge of dealing with the large volume of data generated by such searches.

- **Accessing NCBI Databases**: The `Bio.Entrez` module allows programmatic access to NCBI's Entrez databases for tasks like searching (`esearch`), fetching full records (`efetch`), and finding related items (`elink`). This enables automated querying and data retrieval (e.g., retrieving GenBank records from the net).
- **Parsing Search Results**: The `Bio.SearchIO` module (and the older `Bio.Blast.NCBIXML` for BLAST XML) is specifically designed to parse output files from various sequence search tools like BLAST, FASTA, and HMMER. It can handle different output formats, statistics, and structures of search results (queries, hits, HSPs, alignment fragments). It provides structured objects for easy extraction of information like program name, query ID, target database, hit IDs, descriptions, E-values, and bit scores.
- **Practical Exercises with Biopython.** An example of similarity search: flavocytochrome b2 (PDB code 1fcb). This enzyme has been shown to be very similar to a phosphoribosylanthranilate isomerase (PDB code 1pii) by detailed *three-dimensional structural analysis*. Obtain the yeast flavocytochrome b2 protein sequence from NCBI Entrez. Compare the sensitivity of BLASTP, FASTA, ParAlign, PSI-BLAST, and HMM-based (HHPRED) approaches for detecting remote homology with *phosphoribosylanthranilate isomerase*.

Conclusion:
Database similarity searching is an indispensable first step in characterizing novel sequences, providing crucial insights into their potential function, structure, and evolutionary past. Heuristic algorithms like BLAST and FASTA, alongside robust statistical measures such as E-values and bit scores, enable efficient exploration of vast sequence repositories. As computational power grows and algorithms mature, advanced methods, coupled with programmatic tools like Biopython, empower researchers to uncover even the most subtle and distant homologous relationships, continually driving forward the frontiers of biological discovery.

Exercises

1. Write a function that takes an amino acid sequence and a threshold T and returns the list of all 3-mers that are either in the sequence or their alignment scores with any one of the former is above T.
2. Write a function that takes a query sequence Q, a 3-mer Z from Exercise 1, a threshold t and a target sequence S and aligns the two sequences according to BLAST algorithm. This means it first aligns the 3-mers similar to Q in the two sequences (assume such 3-mers exist in the two), and then extends both ways (without gaps) until the alignment score drops below t.
3. Try to design a query sequence Q and a target sequence S such that their actual alignment score is high but BLAST does not return S as similar to Q.
4. Repeat Exercise 3 for the reverse case: their overal alignment score must be low but BLAST returns S as a similar sequence.
5. Perform the FASTA algorithm by hand for the example sequences in the Dot Matrix section of Chapter 3.