BioE 131: Intro to Computational Biology Fall 2020

Biological Databases and File Formats

Professor: Ian Holmes

Notes written by Vikram Shivakumar

1 Introduction

File formats used in computational biology vary widely in structure and convention, and are often built for specific use cases by different communities with different needs. Some specific formats (like FASTA, GFF, BED, and SAM) are indicative of the these needs, representing specific types of data in uniform, flatfile-style formats. Other formats are more generic and extensible, applicable in multiple different contexts. These are also often used outside of computational biology (like XML and JSON), and follow specific schemas, or templates for how the data should look.

Databases are another method of storage for biological data. These repositories allow for adding, changing, retrieving, and deleting data, as well as user queries of select data. Often the databases impose a standardized form to avoid "dirty data", ensuring a level of quality and consistency across the repository. Databases can also be linked to other databases.

1.1 Databases vs Files

Why use databases at all? Why not just use files to store all data?
One reason is that sometimes file formats can vary across filesystems, which can cause problems for data portability. Data validation also becomes more difficult with data stored in files. Also, linking and indexing data can be difficult across files in a filesystem.
Why not just use databases and no files?
Lots of bioinformatics code and programs still use files to read and write data, and porting over to databases would be too difficult. Also, when running analyses of data, it is often quicker to start with files, and files are often easier to interpret and manually examine.

2 Databases

2.1 Relational vs non-Relational Databases

Relational databases store data that are related to each other in tables. Each table has multiple columns according to a **schema**, which represent attributes of the data. These tables can be connected together by **relational operators**, which specify links through keys. Accessing data from databases can become impossible to do manually as database size increases, but this can be achieved using the **Structured Query Language (SQL)**. This language uses defined set operations to retrieve data from across tables in a database. Lastly, relational databases can be built and accessed through Relational Database Management Systems (RDBMS), like MySQL, Postgres, Oracle, etc.

Databases like MongoDB, CouchDB, and MarkLogic are examples of non-relational databases. These databases follow the idea of "document store", using key-value pairs to point to data stored in documents. These systems can be faster and built larger, and can be useful in scenarios of big data and distributed data.

2.2 Properties of Databases

Relational databases follow the **ACID** properties (where an *operation* is defined as viewing or modifying data):

1. **A**tomicity: operations are all-or-nothing
2. **C**onsistency: operations always leave a system in a valid state
3. **I**solation: operations do not affect each other and thus can be concurrent
4. **D**urability: operations are robust to system failures

Another set of properties is **CAP**:

1. Consistency - each operation returns the most recent result
2. Availability - each operation results in a response
3. Partition-tolerance - operations still occur even if data is lost between partitioned nodes

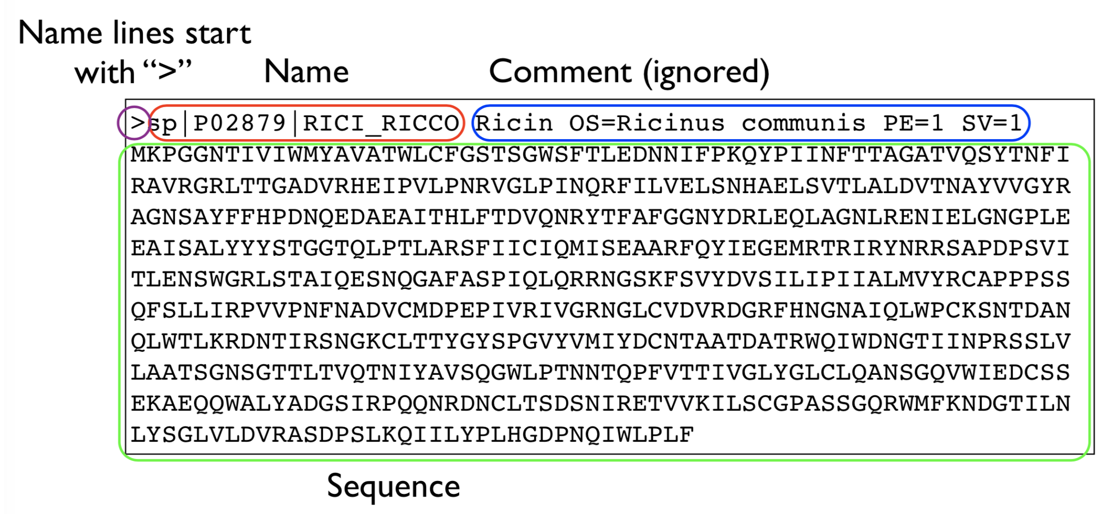
According to Brewer's theorem, a database cannot have all three of the CAP properties. For example, considering a network with two nodes (separate partitions). If there is a failure between nodes (data cannot be sent between nodes), then either to preserve partition-tolerance, either responses to requests have to be halted (no availability), or possibly outdated responses need to be sent (no consistency).

3 File Formats

Now let's take a look at a few different file formats that are commonly used in bioinformatics.

3.1 FASTA and FASTQ

FASTA files are one of the most common filetypes in biology. They are used to store biological sequence data, for example DNA, RNA, or protein sequences. FASTA files are an example of a flatfile, and have a simple structure:



```
>sp|P02879|RICI_RICCO Ricin OS=Ricinus communis PE=1 SV=1
MKPGGNTIVIWMYAVATWLCFGSTSGWSFTLEDNNIFPKQYPIINFTTAGATVQSYTNFI
RAVRGRLTTGADVRHEIPVLPNRVGLPINQRFILVELSNHAELSVTLALDVTNAYVVGYR
AGNSAYFFHPDNQEDAEAITHLFTDVQNRYTFAFGGNYDRLEQLAGNLRENIELGNGPLE
EAISALYYYSTGGTQLPTLARSFIICIQMISEAARFQYIEGEMRTRIRYNRRSAPDPSVI
TLENSWGRLSTAIQESNQGAFASPIQLQRRNGSKFSVYDVSILIPIIALMVYRCAPPPSS
QFSLLIRPVVPNFNADVCMDPEPIVRIVGRNGLCVDVRDGRFHNGNAIQLWPCKSNTDAN
QLWTLKRDNTIRSNGKCLTTYGYSPGVYVMIYDCNTAATDATRWQIWDNGTIINPRSSLV
LAATSGNSGTTLTVQTNIYAVSQGWLPTNNTQPFVTTIVGLYGLCLQANSGQVWIEDCSS
EKAEQQWALYADGSIRPQQNRDNCLTSDSNIRETVVKILSCGPASSGQRWMFKNDGTILN
LYSGLVLDVRASDPSLKQIILYPLHGDPNQIWLPLF
```

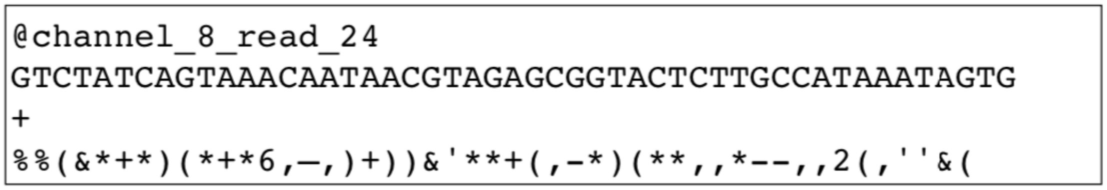
Figure 1: Example of a protein FASTA file

The sequence itself is written using the IUPAC single-letter codes for amino acids and nucleotides. This allows for degenerate codes, e.g. R=[AG],Y=[CT], N=[any nucleotide].

FASTQ files are similar to FASTA files in the representation of sequences, however it also includes an associated quality score for each base (or amino acid). To store this data, non-whitespace ASCII printable characters are used, e.g. "!" or "%", each representing a quality score. Each sequence in a FASTQ file has exactly 4 lines: the name line (always starting with an "@" symbol), the sequence itself, a line with only the character "+", and the sequence of quality-score ASCII characters.

The quality scores in the FASTQ format are generally **Phred quality scores**. These scores are related to the probability P of a base-calling error during sequencing, and are calculated using the formula:

$$Q = -10 \log_{10} P$$



```
@channel_8_read_24
GTCTATCAGTAAACAATAACGTAGAGCGGTACTCTTGCCATAAATAGTG
+
%%(&*+*)(*+*6,—,)+))&'**+(,-*)(**,,*--,,2(,''&(
```

Figure 2: Example of sequencing read in FASTQ format

3.2 Annotation

When working with sequence data, a common problem is identifying regions of significance, such as genes, splice sites, binding sites, or in the case protein sequences, protein domains and motifs. Naturally in these scenarios, a file format that is able to store **annotation** information for a sequence is crucial for many bioinformatics applications.

One format that is commonly used for this purpose is the **General Feature Format (GFF)**. GFF is a type of flatfile which specifies coordinates of various genomic features (such as exons, introns, promoters, etc.) for an accompanying sequence. A similar format, **BED** lists sequence coordinates, but varies slightly in some conventions and formatting.

One source of annotation and sequence data is **Genbank**, a database of all publicly available DNA sequences (it is synced with other databases like EMBL and DDBJ). Large databases like these impose conventional forms on the annotation and sequence files to ensure standardization across the database, which in turn allows bioinformatics tools to work consistently with different datasets. **Swiss-prot** and **TrEMBL** are other examples of curated databases specifically for protein sequences.

Another format for base-resolution annotation of sequences is the **WIG (Wiggle)** format. This format allows for a numerical value to be associated with each base in the associated sequence, a form of quantitative annotation of sequence data. This is similar to the FASTQ format, where a value is associated which each base or amino acid in the sequence.

3.3 Structure

Biological sequences are presented in FASTA (or other formats) as linear strings of nucleotides or amino acids, but in biology, this is often not the case. Especially in the case of proteins and RNA, sequences tend to fold into complex structures, yet another piece of data that needs to be encoded into a file.

The **Protein Data Bank (PDB)** is a databank that stores 3D structures of large molecules like proteins, DNA, and RNA (*Note:* PDB is a databank, not a database, which means the data is not standardized or modified when submitted). Often the 3D structure is determined experimentally using techniques like x-ray crystallography, and submitted to the PDB. The **PDB filetype**, which is used to store this data, contains the atomic coordinates of these crystallographic structures in a column format. This filetype is relatively very old, and still resembles the structure of punch cards, which were once used to store the structural data.

Another common structure to store is **RNA secondary structure**. Due to base pairing within a single RNA sequence, these molecules often form various fold-back structures that need to be encoded in some way. The most common representation of this secondary structure is the **dot-bracket notation**. This notation uses dots to represent unpaired bases, and paired brackets to represent base pairs across the sequence. These brackets can be nested to represent more complex structures like stem-loops.

```
GAGGGCGGUUCCCUCCGCGAGGCGAUGCGCG
<<<<<.....>>>>><<<<........>>>>
```

Figure 3: Example of dot-bracket notation for RNA structure

However, one structure in particular cannot be represented using dot-bracket notation: **pseudoknots**. These structures involve intercalated stem loops, which break the nested, paired nature of dot bracket notation.

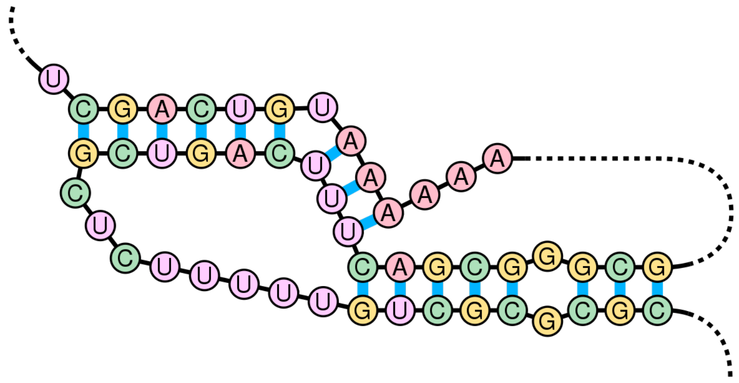


Figure 4: Pseudoknot structure

4 Alignment and Homology

Another common task is determining how similar sequences are by aligning them to each other, and inferring evolutionary relationships from these alignments.

4.1 Alignment

Alignments can be stored in a straightforward way using a format we are already familiar with, FASTA files. Including multiple sequences from a **Multiple Sequence Alignment (MSA)** in a single FASTA, and adding in gaps in each sequence to represent insertions or deletions, is a common way to store short alignments of sequences. However in some cases, there are more efficient methods of storing this data.

Consider for example the task of aligning a million short ($\sim$ 100nt) reads (stored in a FASTQ file) to the human genome ($\sim$ 3 million bp). Using a gapped FASTA file would result in over a million sequences in a list, each with millions of gaps flanking the 100nt read sequence. This is extremely inefficient! The more common approach is to use a **SAM** (or **BAM**) file. SAM files list each read and the alignment position in a reference sequence. That way, only the read sequences and a position need to be stored (assuming the reference is stored in a separate sequence file). A BAM file is the binary equivalent of the SAM file (and is often a smaller file size).

SAM files use a method for efficiently storing alignments called the **CIGAR** format. This format exploits the simple idea that base pairs that are identical to the reference do not need to be stored again. Thus each non-reference sequence is simply stored as a sequence of **Matches (M)**, **Insertions (I)**, or **Deletions (D)** (*Note:* the distinction between an insertion and deletion is arbitrary and depends on which sequence is the ancestor of the other, which is often unknown). The CIGAR format is an example of **run-length encoding**, where multiple identical characters are encoded as a single variable with an associated count, a form of **lossless data compression**.

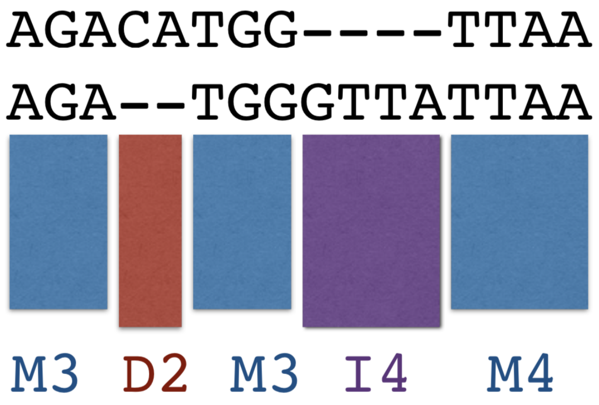


Figure 5: Example of the CIGAR alignment format

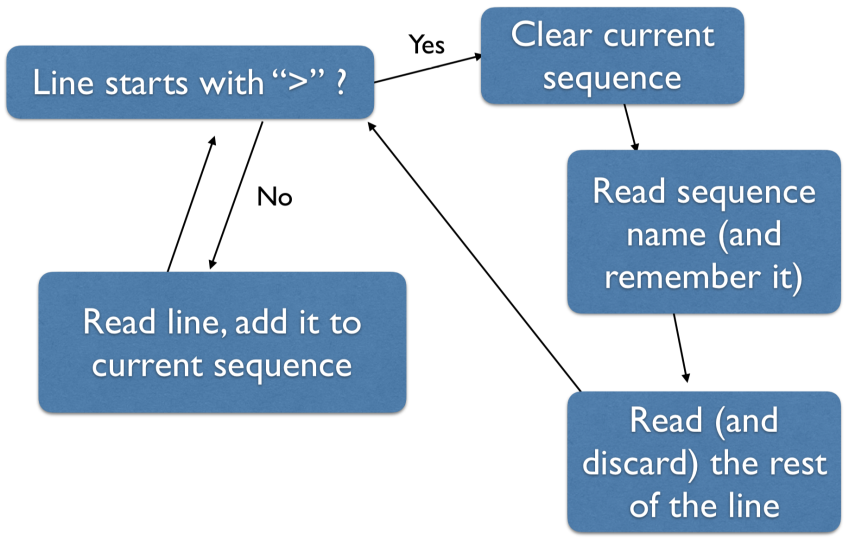
4.2 Evolutionary Homology

Some databases like PFam, BLOCKS, COG, and PhyloFacts contain protein domain alignments. PFam and RFam in particular store alignments in the **Stockholm format**. This format includes the sequence of each alignment entry along with other features of the alignment. Other databases like InterPro contain statistical profiles of protein domains, which can be used to characterize new protein sequences into domains and protein families.

Lastly, another common type of data is the phyogeny, which stores the evolutionary relationships between species and sequences. Phylogenies naturally take the form of trees, which can be stored in a nested format. The **New Hampshire (Newick) format** uses a nested structure to store the tree relationship between different items, along with information on branch lengths, statistical confidence values, etc. The NEXUS format is another format used to store phylogenies, is richer than Newick and contains more information on the sequence alignment and the taxa in the phylogeny.

5 Parsers

Each format has its own type of data, conventions, and standard form, which needs to be parsable in order to be useful in bioinformatic code. Many parser libraries exist in each programming language, so generally no manual parsing is needed. Depending on the format, some parsers may use models like **finite state machines** or data structures like **stacks** (for nested structures) to parse files into the relevant data types. The following is an example state machine to parse a FASTA file:


Line starts with ">" ?
Yes
Clear current sequence
No
Read sequence name (and remember it)
Read line, add it to current sequence
Read (and discard) the rest of the line

6 Summary

File formats are very important in storing the diverse types of data used in computational biology. When thinking about each filetype, some important things considerations include how big the file is, what data is includes, what data does it exclude, and what is the goal of the format. Another important consideration is if the data is being stored *efficiently*. This includes how the format scales with data size, and how compression affects the quality of the data (lossless compression vs lossy compression). In summary, the right file format go a long way when conducting bioinformatic experiments!