Bioinformatics - 16/01/2025

Praanesh Balakrishnan Nair

March 6, 2025

Contents

1 Some Useful Programs

1.1 Reverse Complement

```
from Bio.Seq import Seq


dna = Seq("ATGCCGTA")
print(f"Reverse Complement: {dna.reverse_complement()}")
```

1.2 Measure of Molecular weight

1. 1 Dalton (Da) = mass of a proton/ neutron
2. Mass of the molecule = sum of all the protons
3. Here's how you do it in Biopython

```
from Bio.SeqUtils.ProtParam import ProteinAnalysis
analysis = ProteinAnalysis("VKLFPWFNQY")
mass = analysis.molecular_weight()
print(f"Mass: {mass}")
```

1. Table of the weights of amino acids:

G	A	S	P	V	T	C	I/L	N	D	K/Q	E	M	H	F	R	Y	W
57	71	87	97	99	101	103	113	114	115	128	129	131	137	147	156	163	186

   We have **20 amino acids**, but only **18 integer masses**.

1.3 Isoelectric point

- It's the pH where a molecule has 0 electric charge
- Code to find it in biopython:

```
from Bio.SeqUtils.ProtParam import ProteinAnalysis

analysis = ProteinAnalysis("VKLFPWFNQY")
isoelectric_point = analysis.isoelectric_point()
print(isoelectric_point)
```

1.4 Amino Acid Composition

```
from Bio.SeqUtils.ProtParam import ProteinAnalysis
dna = ProteinAnalysis("ATGCCGTA")

print(dna.count_amino_acids())
```

1.5 Aromaticity

```
from Bio.SeqUtils.ProtParam import ProteinAnalysis

dna = ProteinAnalysis("ATGCCGTA")
print(dna.aromaticity())
```

2 Central Dogma of Molecular Biology

"DNA makes RNA makes Proteins"

2.1 Replication

- Initiation
- Elongation
- Termination

2.2 Transcription

- DNA ⇒ RNA
- It's basically replacing T (Thymine) with U (Uracil)
- **Ribonucleotides**: Adenine, Uracil, Guanine, Cytosine
- To do it in Biopython:

```
from Bio.Seq import Seq
seq = Seq("AGTACACTGGT")
seq_transcribed = seq.transcribe()
print(f"Original: {seq}\nTranscribed: {seq_transcribed}")
```

2.3 Translation

- RNA ⇒ Protein
- Take 3 ribonucleotides (A, U, G, C) at a time
- **Codon**: A triplet of nucleotides

Number of Codons: $4^3 = 64$
Number of Amino Acids: 20

- Codons **code** for an amino acid. In other word, a codon is an encoding of an amino acid.
- A single amino acid can have multiple codons coding for it.
- **Stop Codons**:

UAA UAG UGA

These basically code to stop translation.

- To do it in Biopython:

```
from Bio.Seq import Seq
seq = Seq("AGTACACTGGTG")
seq_translated = seq.translate()
print(f"Original: {seq}\nTranslated: {seq_translated}")
```

3 Antibiotics Sequencing

3.1 What Antibiotic Sequencing is

- A mini protein/ peptide / short string of amino acids which can kill a bacterium, is called an **antibiotic**.
- Sequencing an antibiotic refers to determining its chemical structure.

3.2 Why this is important

- **Drug Discovery**: Drugs like penicillin are life saving substances and they are derived from microbes.
- **Synthetic Biology**: This is where you modify antibiotics to make them more effective.

3.3 How Antibiotics are Sequenced

3.3.1 Mass Spectrometery

- You break down an antibiotic into ions.
- Ions are now passed through an electric field.
- The time taken for each ion tells us the mass of each ion (lighter ions move faster, which heavier ions move slower).
- The $\frac{mass}{charge}$ ratio is calculated for each ion.
- Every time this ratio peaks, you know that a fragment/subpeptide has passed by (and not just small ions).
- These peak values are called a **spectrum**, and scientists use this to reconstruct an antibiotic.

1. Theoretical Spectrum
    - Mass of every possible sebpeptide, plus 0 and the mass of the peptide

eg. Peptide Given = LNEQ Spectrum:

L	N	Q	E	LN	NQ	EL	QE	LNQ	ELN	QEL	NQE	LNQE
..	..	..	..	..	..	..	..	..	..	..	..	..

So you're given with something like `[0, 97, 99, ... 497]`.

2. Noisy Spectra
   - **False mass**: Present in Experimental Spectrum, missing in theoretical spectrum
   - **Missing mass**: Present in theoretical spectrum, missing in experimental spectrum
   - **Score**: Number of masses common in both spectra.

4 Cyclopeptide Sequencing problem:

Given a theoretical spectrum, find out the peptide.

4.1 Brute Force Cyclopeptide Sequencing:

- The mass of the entire peptide is usually known.
- Algorithm:
  1. Generate all peptides with given mass.
     - Say it's 1322. Find all 1-mers, 2-mers, 3-mers ... k-mers that sum up to 1322
  2. Form the theoretical spectrum for each and every k-mer you generated
  3. Look for matches with given spectrum.
- You may not get the old peptide back, because there can be different amino acids with the same mass, and moreover, you can have different **combinations** of amino acids with same mass of the original peptide.

4.2 Branch-and-Bound Algorithms

Say this was the spectrum given:

0	97	97	99	101	103	196	198	198	200	202	295	297	299	299	301	394	396	398	400	40

1. Find the amino acids whose weights lie in the spectrum.

G	A	S	P	V	T	C	I/L	N	D	K/Q	E	M	H	F	R	Y	W
57	71	87	97	99	101	103	113	114	115	128	129	131	137	147	156	163	186

(Let's take the first 4 1-mers)

P
V
T
C

1. Now make all 2-mers out of these 4 1-mers. Basically add all 18 amino acids to each 1-mer

PG	PA	PS	PP	PV	PT	PC	PI/PL	PN	PD	PK/PQ	PE	PM	PH	PF	PR	PY	PW
VG	VA	VS	VP	VV	VT	VC	VI/VL	VN	VD	VK/VQ	VE	VM	VH	VF	VR	VY	VW
TG	TA	TS	TP	TV	TT	TC	TI/TL	TN	TD	TK/TQ	TE	TM	TH	TF	TR	TY	TW
CG	CA	CS	CP	CV	CC	CC	CI/CL	CN	CD	CK/CQ	CE	CM	CH	CF	CR	CY	CW

1. In each of these 2-mers, find which lie in the given spectrum

PG	PA	PS	PP	**PV**	**PT**	**PC**	PI/PL	PN	PD	PK/PQ	PE	PM	PH	PF	PR	PY	PW
VG	VA	VS	VP	VV	VT	VC	VI/VL	VN	VD	VK/VQ	VE	VM	VH	VF	VR	VY	VW
TG	TA	TS	TP	TV	TT	TC	TI/TL	TN	TD	TK/TQ	TE	TM	TH	TF	TR	TY	TW
CG	CA	CS	CP	CV	CC	CC	CI/CL	CN	CD	CK/CQ	CE	CM	CH	CF	CR	CY	CW

And now we have:

PV
PT
PC

1. Now make all 3-mers out of these 3 2-mers. Basically add all 18 amino acids to each 2-mer

PVG	PVA	PVS	PVP	PVV	PVT	PVC	PVI/PVL	PVN	PVD	PVK/PVQ	PVE	PVM	PVH
PTG	PTA	PTS	PTP	PTV	PTT	PTC	PTI/PTL	PTN	PTD	PTK/PTQ	PTE	PTM	PTH
PCG	PCA	PCS	PCP	PCV	PCT	PCC	PCI/PCL	PCN	PCD	PCK/PCQ	PCE	PCM	PCH

1. In each of these 3-mers, find which lie in the given spectrum

PVG	PVA	PVS	PVP	**PVV**	PVT	PVC	PVI/PVL	PVN	PVD	PVK/PVQ	PVE	PVM	PVH
PTG	PTA	PTS	PTP	PTV	PTT	PTC	PTI/PTL	PTN	PTD	PTK/PTQ	PTE	PTM	PTH
PCG	PCA	PCS	PCP	PCV	PCT	PCC	PCI/PCL	PCN	PCD	PCK/PCQ	PCE	PCM	PCH

4.3 Leaderboard Cyclopeptide Sequencing

(work in progress)

5 Sequence Alignment

5.1 Why Align Sequences?

- You can establish the following relationships:
    1. Functional Relationship
    2. Structural Relationship
    3. Evolutionary Relationship

5.2 Types of Alignment

5.2.1 Global Alignment

1. What it is
   - Align all letters from query and target
   - Sequence must be closely related/similar
   - Example: **Needleman-Wunsch**
2. How it works
   (a) Initialization
   - Say we have two sequences $ATGCT$ and $AGCT$
   - Among these two sequences, if the lengths of the sequences are m and n, then make a matrix of size $(m+1)\mathbf{x}(n+1)$

		A	**T**	**G**	**C**	**T**
A						
G						
C						
T						

   (b) Matrix Filling
   Fill the matrix such that
   - 1 = Match (added to diagonal element only)
   - -1 = Mismatch (added to diagonal element only)
   - -2 = Gap

		A	**T**	**G**	**C**	**T**
	0	-2	-4	-6	-8	-10
A	-2					
G	-4					
C	-6					
T	-8					

   - For top/left element you add -2, and for the immediate top-left diagonal element, you add +-1 depending on if it's a match or not
   - The final value of the element, would the maximum of whatever you find

		A	T	G	C	T
	0	-2	-4	-6	-8	-10
A	-2	1	-1	-3	-5	-7
G	-4	-1	0	0	-2	-4
C	-6	-3	-2	-1	1	-1
T	-8	-5	-2	-3	-1	2

   (c) Trackback
   You basically move from the bottom-right corner to the top-left corner. You can do this in 3 ways, and 'moving' means swapping the numbers
   -

3. Another example, where penalties are different

   - 1 = Match (added to diagonal element only)
   - -1 = Mismatch (added to diagonal element only)
   - -1 = Gap

		C	G	T	G	A	A	T	T	C	A	T
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22
G	-2											
A	-4											
C	-6											
T	-8											
T	-10											
A	-12											
C	-14											

4. Code in biopython

```
from Bio import pairwise2

# Given DNA sequences
seq1 = "ATGCTAGC"
seq2 = "ATGCTAGCTAGC"

# Scoring parameters
match = 1
mismatch = -1
gap_open = -2
gap_extend = -2

# Perform global alignment
alignments = pairwise2.align.globalms(seq1, seq2, match, mismatch, gap_open, gap_extend)

# Print best alignment and score
print(pairwise2.format_alignment(*alignments[0]))
```

   (a) `from Bio import pairwise2`

   (b) `pairwise2.align.globalms()`

   (c) `pairwise2.format_alignment(*alignments[0])`

5.2.2 Local Alignment

- Align only the regions with higher similarity i.e. you align only substrings
- This is suitable for more divergent sequences
- Example: **Smith-Waterman**

1. What is is

2. How it works

   (a) Initialization

		A	T	G	C	T
		0	0	0	0	0
A	0					
G	0					
C	0					
T	0					

   (b) Matrix filling

   - Fill the matrix such that
     - 1 = Match (added to diagonal element only)
     - -1 = Mismatch (added to diagonal element only)
     - -2 = Gap
   - But the catch is that if you get a negative value, you make it zero. That's why the initialization is all zeroes. (It was -2, -4, etc. . . , but negative values are truncated to 0)

		A	T	G	C	T
	0	0	0	0	0	0
A	0	1	0	0	0	0
G	0	0	0	1	0	0
C	0	0	0	0	2	0
T	0	0	1	0	0	3

   (a) Traceback

		A	T	G	C	T
	0	0	0	0	0	0
A	0	1	0	0	0	0
G	0	0	0	1	0	0
C	0	0	0	0	2	0
T	0	0	1	0	0	3

3. Another example

		G	A	A	T	T	C	A	T
	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	1	0	0
C	0	0	0	0	0	0	1	0	0
T	0	0	0	0	1	1	0	0	1
C	0	0	0	0	0	0	2	0	0
A	0	0	1	1	0	0	0	3	0
T	0	0	0	0	2	1	0	0	4
G	0	1	0	0	0	0	0	0	0

4. Code in biopython

```
from Bio import pairwise2

# Given DNA sequences
seq1 = "TGTGACTA"
seq2 = "CATGGTCA"

# Scoring parameters
match = 1
mismatch = -1
gap_open = -2
gap_extend = -2

# Perform local alignment (Smith-Waterman Algorithm)
alignments = pairwise2.align.localms(seq1, seq2, match, mismatch, gap_open, gap_extend)

# Print best local alignment and score
print(pairwise2.format_alignment(*alignments[0]))
```

5.3 Longest Common Subsequence Problem

*

6 Genome Assembly

It's the process of getting back a genetic sequence, using numerous short sequences called *reads*.

6.1 Ideas and Efforts

6.1.1 Genome Wide Association Studies (GWAS)

- You identify variations/mutations associated with a disease or the risk of getting a disease

6.1.2 Next-Generation Sequencing

- This is where you try to sequence DNA and RNA, by minimizing time and cost required.
- For instance, linearly sequencing isn't quick and cost-effective

6.1.3 Sanger Sequencing

- Up until early 2000s, this was used to sequence the genomes of many mammals.

6.1.4 Illumina

- It's a machine that came in the late 2000s that reduces the cost of sequencing a human genome from $3B, to $10K

6.2 Why it's a big deal

- In 2010, Nicholas Volker's genome was sequenced and he became the first human saved because of sequencing. He had many surgeries done and this thing helped a lot

6.3 The Procedure

- The genomes you have are like a stack of multiple copies of a particular newspaper. Let's say you blast all of them into a bunch of pieces.
- Each copy would have blasted differently.
- Say we're looking for a phrase "An apple a day, keeps the doctor away".
- If the piece of one newspaper has the words "An apple a day, keeps", the piece of another newspaper has the words "day, keeps the doctor away", you can find the similarity between these pieces "day, keeps".
- One piece contains whatever was before this phrase, and another piece contains whatever was after. So you now have found the complete sentence.
- The same thing is happening with genomes too, just that each "piece" corresponds to a k-mer, and using a random order of k-mers, you'll have to find the original sequence.
- Each "piece" is knowns as a "read".

6.4 String Reconstruction

6.4.1 By Brute Force

- The procedure mentioned above, just that one whole newspaper is a sequence, and each piece is a k_{mer}
- Eg. Given 3-mers {AAT, ATG, GTT, GTT, TAA, TGT}, find the string:
  - Find the starting 3-mer, by looking at only the first two characters of all the 3-mers, and checking which 3-mer doesn't end with these two characters. It's TAA, because there's no k-mer starting with TA.

6.4.2 As Hamiltonian problem

- Given all reads are 3-mers (Example)

1. Form a graph of these 3-mers such that the suffix (last 2 characters) of one node, is the prefix (first 2 characters) of the next node
2. Hamiltonian path is the path where each and every **node** is visited only once Issues:
   - You can have multiple answers

```
def main():
    k_mers = ['AGC', 'TAG', 'CTA', 'ATG', 'CGA', 'AGC', 'GCG', 'TGC', 'GCA', 'GCC', 'CAA', 'A
    hamiltonian_problem(k_mers)

def hamiltonian_problem(k_mers):
    start = None
    for x in k_mers:
        for y in k_mers:
            if x[:1] == y[1:]:
```

```
                break
        start = x



if __name__ == "__main__":
    main()
```

6.4.3 As Eulerian Problem

1. Eulerian path is the path where each and every **edge** is visited only once
2. Then you find the the Debruijn Graph for this Issues:
    - Assumption that every read is a k-mer, is unrealitic
    - Assumption that every read is errorless
        - If there are errors, you'll get "bubbles" in the De Bruijn graph
    - Unknown multiplicity k-mer