15-02-2022

Programming Project 03

Likelihood Protein Identification via k-mer Genetic Sequence Assembly

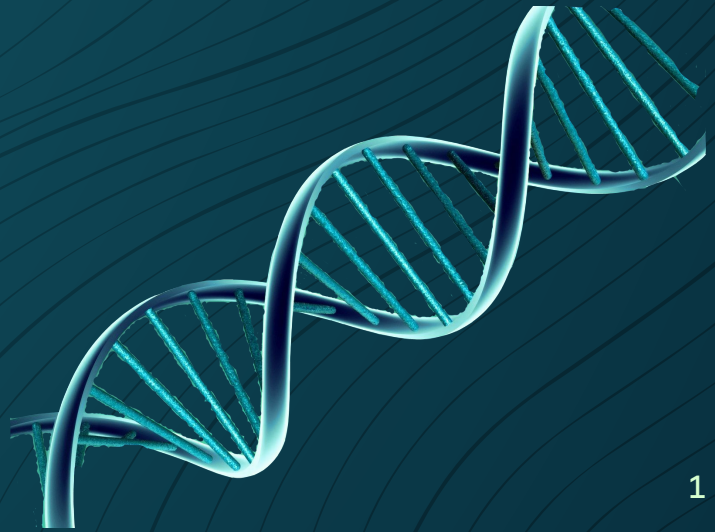
Presentation Team:

Faiza khurshid

Kriti Amin

Elena Xerxa

Sarah Imani



Genetic Sequence introduction:

Molecular biologists and biochemists often utilize strands of DNA to perform experiments. This requires knowing the exact genetic sequence of the DNA they plan to use or modify.

In order to sequence a large strand of DNA, one must first break into smaller fragments, determine the base-pairs of these fragments, and reconstruct the original sequence from the individual reads.
This is known as **shotgun sequencing.**

Genetic sequence introduction conti…:

this method of sequencing large strands of DNA revolutionized the field and made it possible to sequence the human genome.

k-mer

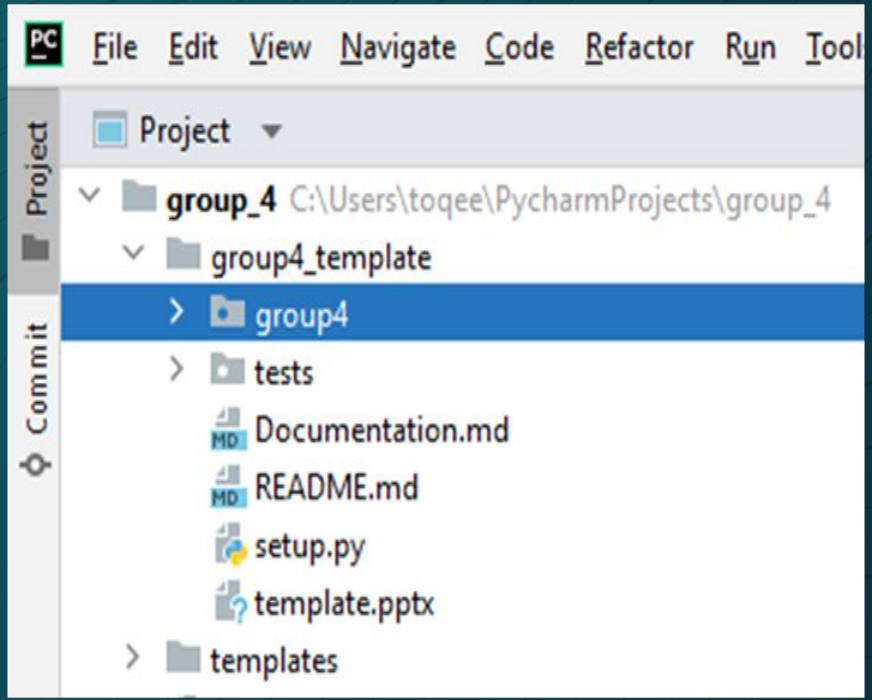
k-mer is substrings of length k contained within biological sequence.

Use in context of computational genomics and sequence analysis in which k mer used as nucleotides (A,T,G andC)
k-mer capitalized upon assembled DNA sequence

k-mer analysis highly use in metagenomic (detect disease)

Project Package:

The **group4 package** a pipeline for assembling fixed-length DNA strand reads into a complete sequence and predict which protein the completed strand may represent



Steps

The package is used to:

1-.Perform de novo sequence assembly on a set of DNA strands (k-mers).

De novo sequence assembly is a technique for assembling shorter fragments of DNA into a longer one through the use of multiple alignments.

2-The genetic sequence undergoes the process of **transcription** and **translation** to form proteins

3-Predict which protein the assembled DNA strand may represent.

GUI:

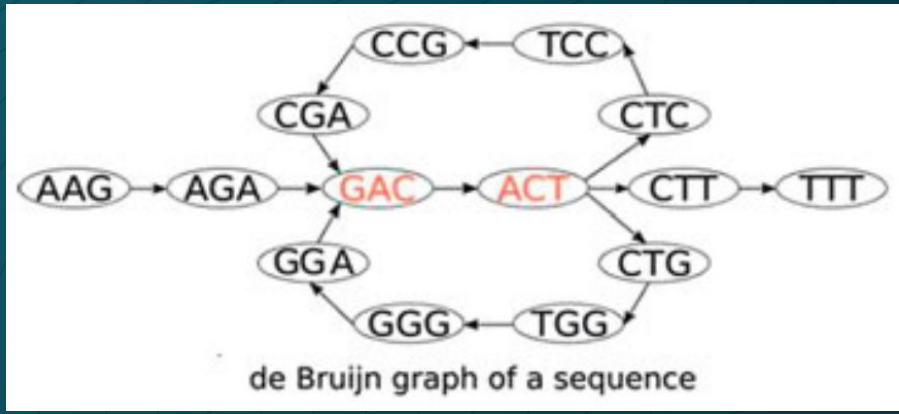
Construct a **web interface** that allows one to upload a file of DNA strands (such as a FASTQ file) and get a list of possible proteins

Outline

1. Strands, Assemble!
2. Central Dogma

Sequence assembly method

- De Bruijn Graph-Based method
- Approach
  - nodes - all possible kmers of reads
  - edges - overlapping kmers
  - find eulerian path
- Suitable for large sets of short reads
- Why not De Bruijn?
  - Complex graph structures
  - Multiple eulerian paths
  - Do not preserve positional information



de Bruijn graph of a sequence

Ref: Narimani, Zahra & Hosseinkhan, Nazanin. (2013). De Novo Assembly Algorithms. doi:10.1007/978-1-4614-7726-6_4.

De novo sequence assembly

- Greedy algorithm
  - Pairwise alignment
  - Two reads with largest overlap
  - Merge and repeat
- Current technologies
  - SSAKE - Short Sequence Assembly by progressive K-mer search and 3’ read Extension
  - PERGA - Paired-End Reads Guided Assembler

- Ref:
- Narimani, Zahra & Hosseinkhan, Nazanin. (2013). De Novo Assembly Algorithms. doi:10.1007/978-1-4614-7726-6_4.
- Khan AR, Pervez MT, Babar ME, Naveed N, Shoaib M. (2014). A Comprehensive Study of De Novo Genome Assemblers: Current Challenges and Future Prospective. doi: 10.1177/1176934318758650. PMID: 29511353; PMCID: PMC5826002.

Module structure

- Module - sequence_assembly
- Class FatstaTools
  - deals with fasta files
  - fasta, fastq
- Class MSA
  - perform multiple sequence alignment
  - Smith-Waterman algorithm (modified)
- Class Assembly
  - perform sequence assembly
  - de_novo_assembly

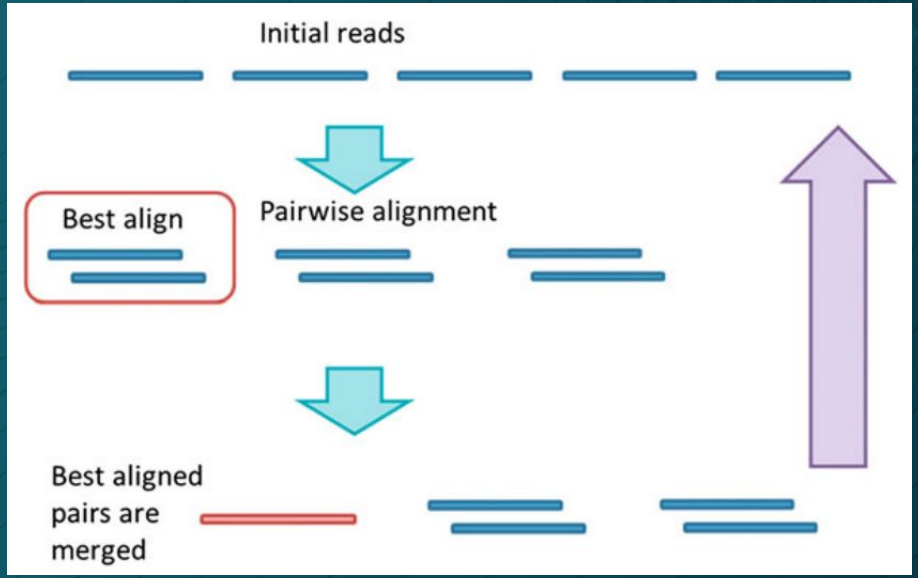


Image: Narimani, Zahra & Hosseinkhan, Nazanin. (2013). De Novo Assembly Algorithms. doi:10.1007/978-1-4614-7726-6_4.

- Challenges
  - Assembly method
  - Minimize runtime

- Limitations
  - Specific input
    - ungapped sequences
    - overlapping ends
  - Large set of sequences
    - lengthy sequences
    - long runtime

Protein Synthesis

- Module - gene_finder
- Complementary DNA
  - dna to cdna
  - reverse strand
- Transcription
  - dna to mrna
  - rna splicing (remove introns)
  - filter proteins - threshold
- Translation
  - mrna to amino-acid sequence
  - List and dataframe

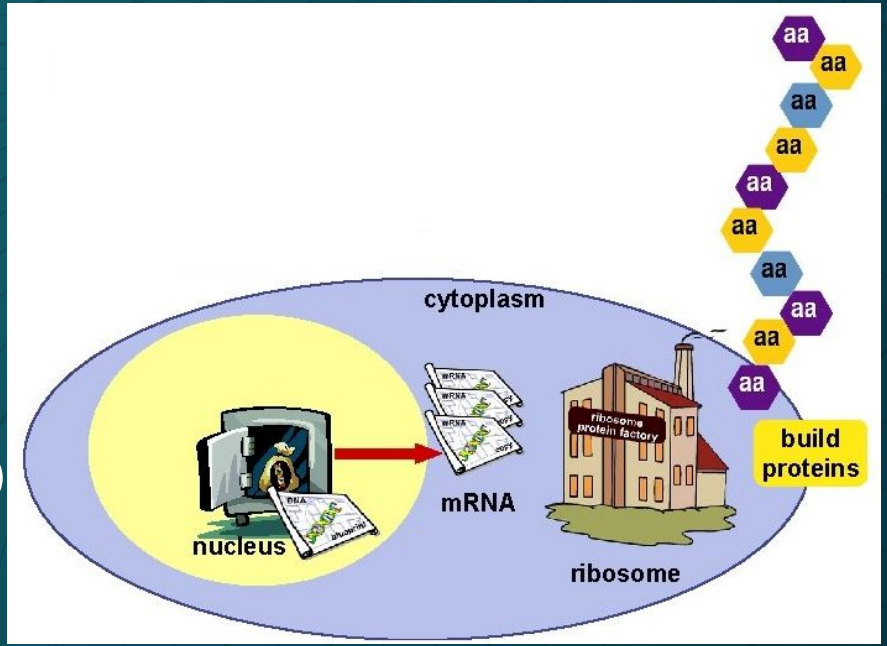


Image: https://slidetodoc.com/protein-synthesis-making-proteins-2009-2010-bodies-cells/

Outline

1. Strands, Assemble!
2. Central Dogma
3. Protein Prediction

Protein Prediction

K-mers

3’-ATGGTCCCAAT
GGTCCCAATG
CCCAATGTTA
AATGTTATCGG -5’

DNA

3’-ATGGTCCCAATGTTATCGG-5’

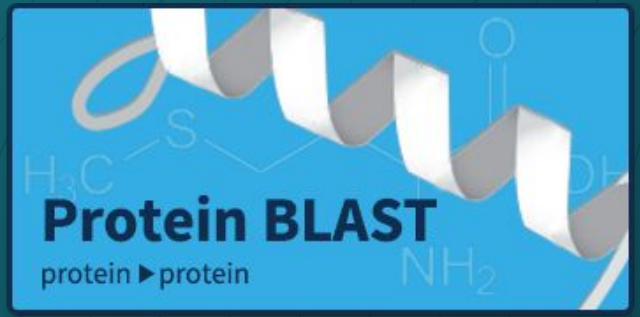
5’-UACCAGGGUUACAAUAGCC-3’

mRNA

ORF 1
ORF 2
ORF 3

Aminoacid sequence 1

Aminoacid sequence 2



?

Protein Prediction

Blast.py

Build-up the query

Send the request

Get the Request ID (RID)

Ask request Status

Get and Download the results

Protein Prediction

Build-up the query

Utils.py

```
URL_endpoint = "https://blast.ncbi.nlm.nih.gov/Blast.cgi?"
PUT_Request = "CMD=put&“

Program = "PROGRAM=blastp&“
Database_PDB = "DATABASE=pdb“
```

Default filters/parameters

Protein Prediction

Send the request

- Import **request** module

```
p = requests.put(PUT_query)  # Submit the request to the BLAST site
```

- **Do not contact the server more often than once every 10 seconds.**

Protein Prediction

Get the Request ID (RID)

```
...
<!--QBlastInfoBegin
    RID = 0CS8M4J6013
    RTOE = 26
QBlastInfoEnd
-->
</form>
...
```

```python
def extract_attribute(p, rid_attr:str=RID)→str:

    """ Will go line by line through a file, searching for mentions of
     the rid_attribute and extract/return the value associated with it
    :param p: str
        -response object generated by querying the server
    :return: str.
        -attribute found in the response file (Default is "RID =").
    """
    put_response = p.text  # get html/text file
    for line in put_response.splitlines():
        if rid_attr in line:
            attribute_value = re.sub(rid_attr, "", line)
            attribute_value = "".join(attribute_value.split())
            return attribute_value
```

Protein Prediction

Ask request Status

```
def check_request_status(rid:str)→str:

    """ Check the request status of a submission.
    :param rid: str
        -Request ID returned when the search was submitted
    :return: str
        -Status of the research
    """
    url_submit = URL + 'CMD=Get&FORMAT_OBJECT=SearchInfo&RID=' + rid
    # Submit the request of the status:
    submit_request = requests.get(url_submit)
    query_status = extract_attribute(submit_request, "Status=")
    return query_status
```

Do not poll for any single RID more often than once a minute.

Protein Prediction

Get and Download the results

```
def get_results(rid:str, response_format:str="html"):

    GET_query_head = URL + CMD=get&RID= + rid

    if response_format == "json":
        GET_query = GET_query_head+rid+ "&FORMAT_TYPE=JSON2

    s = requests.get(GET_query)

    if not s.ok:
        logger.error(f"{GET_query} returned bad status code:
{s.status_code}")
        return
    else:
        logger.info(GET_query)
        return s
```

Save html file

```
with open(path, "w") as
file_handle:
    file_handle.write(s.text)
```

Save json file

```
with open(path, 'wb') as
file_handle:
 file_handle.write(s.content)
```

Protein Prediction

Main Issues:

First request very fast!

Subsequent request very slow…

- Filter the query
- Run over weekend
- Cache files

- Ask for one json files…get two zipped files!

```
from zipfile import ZipFile
zf = ZipFile(filepath)  # convert the zip file to a python object
for filename in zf.namelist():
    if "_" in filename:     # Take only the file with the results
     …
```

Protein Prediction

Wrapper function

```
def Blast_orfs(OrfList:list, filename:str="temporary", program:str = Program, database:
str = Database_PDB, filters: str = "", email: str = "", file_type="html",
keep_files=True):
    n = 0
    dict_matches = {}
    for orf in OrfList:
        n += 1
        file_name = "orf" + str(n) + "_" + filename
        list_matches = Blast_sequence(orf, filename=file_name, program=program,
database=database, filters=filters, email=email, file_type=file_type,
keep_files=keep_files)
        dict_matches[orf] = list_matches
        time.sleep(10)  # Do not contact the server more often than once every 10 seconds.
    logger.info("The job is complete for all the sequences!")
    return dict_matches
```

Outline

1. Strands, Assemble!
2. Central Dogma
3. Protein Prediction
4. GUI

GUI

- An upload button to upload a file of DNA strands
- Collapsable buttons that when pressed will show:
  - Assembled DNA sequence
  - mRNA sequence
  - A table of possible amino acid sequences and the proteins they represent
- A button to download the table of proteins as an Excel file or CSV

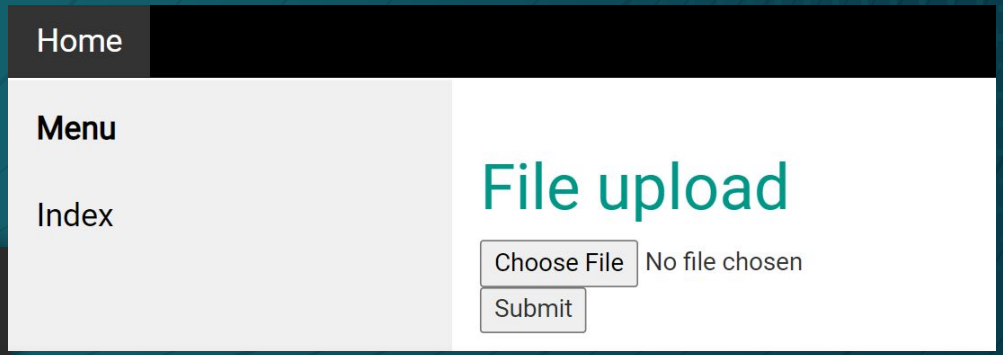
Flask Routing

```python
app = Flask(__name__)
app.secret_key = b'_5#034587Q564482c]/'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MAX_CONTENT_PATH'] = 16 * 1024 * 1024
ALLOWED_EXTENSIONS = {"fasta", "txt", "fastq"}


def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route("/")
def home():
    return render_template('index.html')


@app.route('/upload')
def upload():
    return render_template('upload.html')
```

```python
import os
from pathlib import Path

home_dir = Path.home()
PROJECT_DIR = home_dir.joinpath(".project3", "Group4")
DATA_DIR = PROJECT_DIR.joinpath("data")
UPLOAD_FOLDER = os.path.join(DATA_DIR, 'uploads')
DATA_CACHE = os.path.join(PROJECT_DIR, "data")
os.makedirs(DATA_CACHE, exist_ok=True)
```

Upload File

```python
@app.route('/uploader', methods=['GET', 'POST'])
def upload_file():
    file = request.files['file']
    if request.method == 'POST':
        if file.filename == '':
            flash('No file selected')
            return render_template('upload.html')
        elif file and allowed_file(file.filename):
            flash('File uploaded successfully')
            return render_template('upload.html')
        else:
            flash('Not allowed')
            return render_template('upload.html')
    return redirect(url_for('home'))
```




```html
<form action = "/uploader" method = "POST" enctype = "multipart/form-data">
   <input type = "file" name = "file" />
   <input type = "submit"/>
</form>
```

```jinja
<div class="container">
    {% for message in get_flashed_messages() %}
    <div class="alert alert-warning">
        {{ message }}
    </div>
    {% endfor %}

    {% block page_content %}{% endblock %}
</div>
```

Collapsable Buttons

```python
elif file and allowed_file(file.filename):
    filename = secure_filename(file.filename)
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
    txtfile = open(str(os.path.join(UPLOAD_FOLDER, 'filename.txt')), "w")
    txtfile.write(filename)
    txtfile.close()
    flash('File uploaded successfully')
    result1 = get1()  # Assembled DNA
    obj=Translate(dna=result1)
    result2=''.join(obj.f_mrna)  # mRNA seq
    result3=''.join(obj.r_mrna)  # Reversed mRNA seq
    aa_seq= obj.proteins
    txtfile = open(str(os.path.join(UPLOAD_FOLDER, 'ORFList.txt')), "w")
    txtfile.write(str(aa_seq))
    final_dict = Blast_orfs(obj.proteins)
    obj.proteins_table['predicted_proteins'] = obj.proteins_table['amino_acid_sequence'].apply\
        (lambda x: final_dict[x])
    result4=obj.proteins_table
    return render_template('upload.html', rs1=result1, rs2=result2, rs3=result3,
                           tables=[result4.to_html(classes='data', header="true")])
```

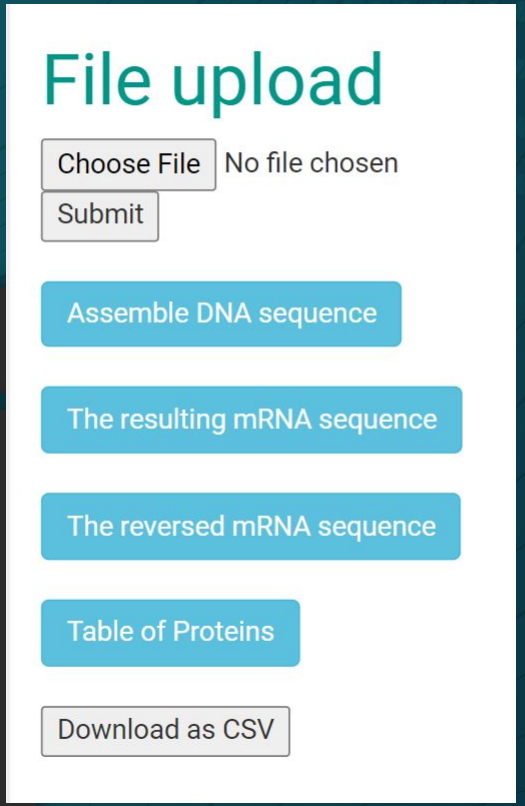
```html
<a href="#btndna" class="btn btn-info" data-toggle="collapse">Assemble DNA sequence</a>
<div id="btndna" class="collapse">
    <span>{{ rs1 }}</span>
```

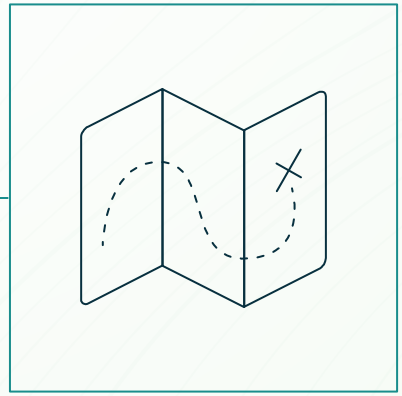
Collapsable Buttons

```
return render_template('upload.html', rs1=result1, rs2=result2, rs3=result3,
                       tables=[result4.to_html(classes='data', header="true")])
```

```
    <a href="#demo4" class="btn btn-info" data-toggle="collapse">Table of proteins </a>
<div id="demo4" class="collapse" >
</br>
     <button type="button" onclick="tableToCSV()">
         Download as CSV
     </button>
     <div id="wrapper">
  {% for table in tables %}
     {{ table|safe }}
```

```
#wrapper {
    overflow-x: auto;
    width: 100%;
```





Thanks!

Any questions?