Project 2

March 25, 2021

0.0.1 EECS 730 Project - 2

Steps followed

1. We have fragments (reads) of a dna sequence in a fasta file. Read the fasta file and collect all the reads in to a list.
2. First derive the individual reads that have overlaps between them.
3. Gather such reads together to create a consensus out of them. In this step, we create the shortest path for all the reads that have overlaps between them.
4. Once we have the consensus for the reads from step2, we do this recursively so that we get a final contig which is the shortest path encompassing all the reads without any repetition.
5. We use Pydna for this task. Below features of Pydna were used a. To derive overlaps between reads (multiple) b. To assemble reads and find the shortest path.
6. We use below cases to test the reconstructed sequence a. Reconstructed sequence should cover every read from the input file. We validate every read, if it is a substring of the reconstructed sequence. b. Reconstructed sequence should not have repetitions. We find how many times a read exists in the reconstructed sequence. Each read should exist only once.
7. Write the reconstructed sequence in to an output file in the below format.

“>reconstructed genome sequence ”ouput assembled sequence

Import relevant packages

[1]:
```
# import packages
import os
import Bio
from Bio import SeqIO
from Bio.Seq import Seq
import pandas as pd
import dask.dataframe as dd
from dask.multiprocessing import get
from pydna.common_sub_strings import terminal_overlap
from pydna.assembly import Assembly
from pydna.dseqrecord import Dseqrecord

# Print versions
print('The Biopython version is {}..'.format(Bio.__version__))
```

```
The Biopython version is 1.78..
```

Create the paths for reference files

[2]:
```
# Set the local paths for data
path = r'C:\Users\pmspr\Documents\HS\MS\Sem 4\EECS 730\Bioinformatics\Project␣
↪2\Docs'
reads = os.path.join(path, 'HW2_reads.fasta')
output = os.path.join(path,'sequence_assembly.txt')
```

Important methods

[3]:
```
# This method derive the shortest path for the given list of sequences
def getcontig(seqlist,seq):
    dseq = tuple(Dseqrecord(seq[s]) for i, s in enumerate(seqlist) if i < 9)
    x = Assembly(dseq, limit=49)
    contigs = x.assemble_linear()
    if len(contigs) > 0:
        return contigs[0].seq.watson
```

[4]:
```
# This method verifies if there is an overlap of certain threshold between␣
↪sequences.
# Threshold = 50bp according to project instructions
def compare_overlap(s1, s2):
    overlaps = terminal_overlap(s1,s2, limit=49)
    if len(overlaps) > 0:
        return 'y'
    else:
        return 'n'
```

[5]:
```
# This method derive the contigs for sequences with overlap
def contigs(seq):
    lr = [i for i in range(0,len(seq))]
    contigList = []
    while (len(lr) != 0):
        i1 = min(lr)
        c1 = [i for i in lr if compare_overlap(seq[i1],seq[i]) == 'y' ]
        c1 = sorted(c1)
        contigList.append(getcontig(c1,seq))
        #print(c1)
        lr = list(set(lr) - set(c1))
    return contigList
```

Main logic

[6]:
```
# Gather all the sequences from th input fasta file
seq = []
with open(reads) as genome:
    for line in genome:
        if(line[0].strip() != '>'):
            seq.append(line.strip())
```

```
print('Total number of reads in file is {}..'.format(len(seq)))
seqlist = seq

# Derive assembled DNA sequence from the individual contigs
while (len(seq) > 1):
    contigList = contigs(seq)
    seq = contigList

print()
print('Assembled DNA sequence..')
print(seq[0])

# Write the output to a fasta file
# Delete the output file if exists
if os.path.exists(output):
  os.remove(output)

# Open the output file in append mode.
outputfile = open(output,"a")

# Write the extracted protein sequence to an output file in fasta file format.
outputfile.write(str('>' + 'reconstructed genomic sequence' + '\n'))
outputfile.write(str(seq[0] + '\n'))

# Close the output file
outputfile.close()
```

```
Total number of reads in file is 127..

Assembled DNA sequence..
CCCTGTCTACCACCCAGACTATCGTGTAGTTCTGCCTGTTCCGTAAGTCGTAGATTGCTATCCTGGAAATCATCGTGCTC
AGGATGTTAATATCTAGCGTCCTACGTTACGAGTTGGCAGATGACAGATCGTAGTCGTGGTAAGGGGCATTGCCGCTTGT
GACCCAGTTCGCGTGCCTAGCAGCACTCCAAAATAAAGTTTACAGTACCGTCCGGACGGCAGAACTGTCCTCTAGATCGT
CCTAACGCCTTAGTCGAATCCCTTGCCGTCGGTAACCACTGAATAAACTACGCGTTAGGACTTTGTCAGACGCGAGGAGC
TAGTAGGAGGACAAATCAGCAAACGACCCTGAATTGAACAATGTGAGTAGGTATAACTGTGCTTGTATGACGTCCCGTTC
GGTCGTTCTTGAGCAACTTCGGCCAGTGCATGCTATGGGGGAAGCTATGAATTCTATGTTGGAACTTGGGCCCGGCATAG
TAGTTTATGCCTGTGGACCGGTGTTGAGTGTATCTGCTGGACCCCGGCGCGTTCACCTGTCCACATCTAATCCAAACATA
TACTATTGGTATTTGAGCGTCTCACAACGACATCGACTGGTATTAGACACCTACCAGGAACAACCAATCGGTTTAGATGA
CGCACAGCCACGGACAGCCTCTGTTGCTTGAGCAGTCCCAAAGTGCGTACCTGAAGCCTGCCAAAACGTAGCCTAGGCAA
ATGCCCGTCGTCTTGCTCATAACTCCTTGGGACTGGCGTATCCATAAATAATCCATTCGATTCCTTGAGAGTTCCACATT
AGAGACTTATCCATCGAGGATCAGGCCAAATCCGCGAGACCCGACCGAGATCAAGTATAACTCATTACGCGTGGTGTGGT
TGCGGCCCACCCTTATCGTGAGCCAGTTGTTGGATATACCCCTGGGCGGGCCTAAAGCTCCGCAACGAACACCCCCTCCG
CTGTGTCTGGTCGATTCTGGCTAGCCGCTCCGTTTGGGTAACAAATCACAGGACGCCATGGATTGCCTCTTAAGTCTGGC
CAGTCGGGATGTCCTGCAGCCCGGGTCACTTCTCTGGTACCCTCTTGGCATAACTTCTTCAAATTTAGAGTTTAATGTTT
CGGGTGAGCTGCATACTGTGATGGGGGGTACTTGGCGTCAAGCGCCACCTTTAGTAGTACTCGAAAAGGCTCATGGTAAA
CCGTGCATACGTTCTAAGGTTCCCCGCTATGGACTGGACCGACCACGAAGTGGTGGATCATGAGACGTACCTGCTTAAGT
CGGTGATTGACGTCACACCTCTCGCGCCCATAGCGAAGCTTATGTACTAAACCCCTTAGTGTTAAGTCCTTACATCTGTT
ATGTCTATTGGAGGACAAGGGGTGTACGCTGCACAGAGCCTTCTTCAGGTAGGAAGAATACAAAATGCCTTTTTCGACAC
```

```
GTATAAGCCGACGAGAGTAACACTGATATCAGCGAAAAATGGGCCCGGGTGTAGAGCGATGTATTTGCTGTTTTATGTGT
CCAGGCAACTCCTAACGTCGTTAAGAAGCACCTTTCTTAGATCCACCGACCCGCGCTGCCTACAATGAGCACTTGTGCTT
CTCATATTTAATACTTCGCATACCTAGAAACCACAAGTAGCTGACCGACTAACGCAACGGCTTCGATGATAAAGTATTGA
CCTTTCGCTTTTTTGACATACTTCCCCGTCACCTGCGATCGGGCCCGGTGTGTTCATATACGATGCCTCTCCACTTGTCG
ACAAGCCAGTCACTATGTAAGCGAACCACCATAATTGATCGACGATAAAGTGACGCGTCCATGCTCATGTATTTATATGA
CGGCCAAAAATGGAGATATTATAGTCGACCAAGTATTGGCGTCGAACAACCGCGCCCTGCAGAATCCCAAGATTCGCCAG
GCGGCGAACGAGGCCTACGGGCAACGGGTTATACTTAGCTGCAACCAACGCCTTTCCACATGTTTGAGAA
```

Testing

[7]:
```
# Check if all the reads are part of assembled sequence
from pydna.common_sub_strings import common_sub_strings

# Check if each read is a substring of the assembled sequence
outliers = [i for i in range(0, len(seqlist)) if␣
 ↪len(common_sub_strings(seqlist[i],seq[0], limit=99)) == 0]
print('Number of reads that are NOT part of assembled sequence {}..'.
 ↪format(len(outliers)))

# Check if each read is a repeated substring of the assembled sequence
outliers = [i for i in range(0, len(seqlist)) if␣
 ↪len(common_sub_strings(seqlist[i],seq[0], limit=99)) > 1]
print('Number of reads that are repeated in assembled sequence {}..'.
 ↪format(len(outliers)))
```

```
Number of reads that are NOT part of assembled sequence 0..
Number of reads that are repeated in assembled sequence 0..
```