

ADVANCED METHODS IN BIOINFORMATICS

MASTER'S IN APPLIED COMPUTER SCIENCE AND ENGINEERING

VRIJE UNIVERSITEIT BRUSSEL

ASSIGNMENT-2 (VUB 4018221FNR / ULB INFO-F439)

2020-2021

LECTURER: Matthieu Defrance

TABLE OF CONTENTS

S.NO.	TITLE	PAGE NO.
1	INTRODUCTION	2
2	MAPPING SOFTWARE ALGORITHM	2
3	CODE EXPLANATION	2
4	CONCLUSION	7
5	REFERENCES	7

REPORT DONE BY

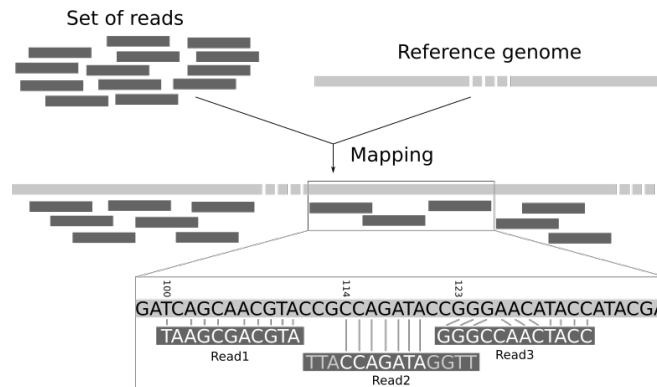
ABHISHEIK KRISHNAGIRI TUPIL RAVIKANTH

0575002

1.INTRODUCTION:

1.1 WHAT IS A READ MAPPING?

The method of aligning reads on a reference genome is known as read mapping. A reference genome and a collection of reads are fed into a mapper. Its aim is to align each read in a collection of reads against the reference genome, allowing for mismatches, indels, and the clipping of some short fragments on the reads' ends.



2.MAPPING SOFTWARE ALGORITHM:

1) “Seed and Extend Algorithm”

I have used the Seed and Extend Algorithm for the Read Mapping Software. The main purpose of using the seed and extend algorithm is because it splits up the complete process into two halves and reduces the computing time. The first step is where the complete Genome file is hashed or broken into possible K-mers and is stored in a form of Hash Table in Dictionary format. In the Dictionary it is stored as a Key: Value pair. In our case the Key is the possible K-mer and the Value is the indices where the corresponding K-mers are present in the genome file. The next step would be to get the short-read sequence and check whether it is present in the dictionary which was created using the genome sequence. Once if the short-read is present in the dictionary, the value (index position) of the corresponding short-read will be seen and then the genome sequence will be extended from there and then will be matched with the read sequence. The Methods which I have used in my code will be explained in the following sections.

3.CODE EXPLANATION:

1) Libraries Imported:

```
# Importing the BioPython library to read the fasta and fastq file and the
→fuzzywuzzy library to find the mapping ratio of the reads to the genome.
from Bio import SeqIO
from fuzzywuzzy import fuzz
```

Figure:3.1.1: Importing the Necessary Libraries.

Libraries in python are pre-defined and are very useful in doing various functions. In our case, I have imported the BIOPYTHON and the FUZZYWUZZY library in order to assist the file parsing and string-matching functions in our use case. The BIOPYTHON library has various functions but we are only using the SeqIO which is used for parsing the FASTA and FASTQ files respectively. The FASTA and FASTQ files contains various information, but we are only in need of the sequence which is present inside it. From FUZZYWUZZY library we are using the FUZZ function which is useful for matching two strings.

2) File accessing and parsing:

```
# This function is used to open the fasta file (Genome Sequence) and return the  
↪ complete sequence in the form of a string.  
def fasta_file(file_name):  
    for record in SeqIO.parse(file_name, "fasta"):  
        fasta_seq = record.seq.upper()  
    return str(fasta_seq)  
  
# This function is used to open the fastq file (Sequence Reads) and return the  
↪ complete sequence in the form of a list.  
def fastq_file(file_name):  
    fastq_list = []  
    for record in SeqIO.parse(file_name, "fastq"):  
        fastq_list.append(str(record.seq))  
    return fastq_list
```

Figure:3.2.1: File accessing and parsing

The **fasta_file(file_name)** function is used to open and parse just the sequence of the Drosophila melanogaster genome sequence. It can also be seen that the output of the function is returned in form of a string. Since we have to index the whole sequence , it is easier to do if it is a string type.

The **fastq_file(file_name)** function is used to open and parse just the sequence of the Drosophila melanogaster read sequence. It can also be seen that the output of the function is returned in form of a list. Since there are 10,000 reads which are present and we have to sequentially pass it through our function , we are using a list here.

We can see how the functions are used to open and parse the sequence and assign to the respective variables.

```
# Storing the genome sequence in a variable in order to access easily.  
genome = fasta_file('chr2L.fa')
```

```
# Reading the fastq (Read sequence) file and storing it in a new variable.  
read_seq = fastq_file("10k_reads.fastq")
```

Figure:3.2.2: Reading the FASTA and FASTQ files and storing it to a variable

Figure:3.2.3: Output of the Genome variable (full output cannot be screenshotted)

Figure 3.2.4: Output of the read_seq variable (full output cannot be screenshotted)

3) Finding all the possible K-mer:

```
# This function finds the possible K-mers of seed size = 10 for the Genome file  
→ and returns the dictionary of all the possible K-mers and their index  
→ position.  
def kmers(sequence, seed_size):  
    hash_map = {}  
    for i in range(len(sequence) - seed_size + 1):  
        kmer = sequence[i:i+seed_size]  
        if kmer not in hash_map:  
            hash_map[kmer] = [i]  
        else :  
            hash_map[kmer].append(i)  
    return hash_map
```

Figure:3.3.1: Finding all the possible K-mers for the Genome sequence.

The **kmers (sequence, seed_size)** function is written in order to find all the possible k-mers for the genome sequence. The first process in the seed and extend algorithm is followed here. First the respective seeds are found. For that this function takes in two arguments, one is the “sequence” which is the genome and the “seed_size” which is the size of the seed to which we want the k-mers to break down to. We are storing the output in the form of a dictionary in order to access it later in the code. In the hash_map dictionary, the output will be stored in a KEY:VALUE pair which in our case will be K-MER: INDEX pair. Once this process is done, we will be assigning the output into a variable to easily reference it later in the code.

```
# Storing the genome hash table in a new variable in order to access it easily.  
genome_table = kmers(genome,10)
```

Figure:3.3.2: Storing the hash map in a variable named genome_table

```
genome_table  
{  
  'CGACAATGCA': [0,  
    458,  
    916,  
    1374,  
    1832,  
    2290,  
    2748,  
    3206,  
    3664,  
    4122,  
    4580,  
    5038,  
    223144,  
    717271,  
    2988514,  
    3118612,  
    4295105,  
    4548076,  
    8868392,  
    10307586,  
    10670268,  
    11374042,  
    15328042,  
    15503890,  
    15590454,  
    16091982,  
    16637592,  
    17722574,  
    18811243,  
    19750674,  
    20845894],  
  'GACAATGCAC': [1,  
    459,  
    917,  
    1375,  
    1833,  
    2291,  
    2749,  
    3207,  
    3665,  
    4123,  
    4583]
```

Figure:3.3.3: Sample output of the genome_table.(full output cannot be screenshotted)

4) Extending and matching it with the reads:

```
# Read Mapping to the Genome Sequence using the seed and extend algorithm and
→it prints the Exact Index from the Genome where the reads are found.
for seq in range(len(read_seq)):
    reference = ''
    short_read = read_seq[seq][:10]
    if short_read in genome_table.keys():
        indices = genome_table[short_read]
        for i in indices:
            reference = genome[i:i+36]
            gg = fuzz.ratio(reference,read_seq[seq])
            print(f'\n The read sequence is {read_seq[seq]} is at the index {i} to
→{i+36} in the genome sequence with a matching score of {gg}')
```

Figure:3.4.1: Code for extending the genome at that instance of index and matching with read.

The above code extract is the final part of the seed and extend algorithm. This is where the read is also broken into 10 character long sequence because it is used to reference with the genome hash map table and retrieve the necessary information(indices) about that short read and then that information is used to map it with the genome sequence. In my case to compare both the string sequences that is the extended genome sequence and the read sequence and to tell how much mapping has been done , I have used the FUZZYWUZZY library's FUZZ RATIO function which compares the two sequences and provides me with the value on how much percentage both the sequences have been matched.

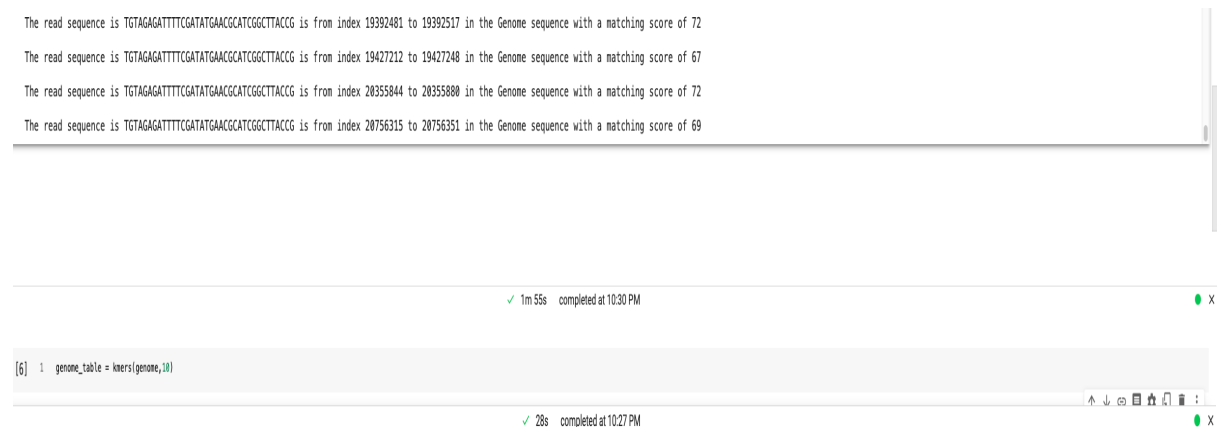
```
The read sequence is CTTTCATTCTCTATCTTATATTACGGCACACACACA is at the index 18946299 to 18946335 in the genome sequence with a matching score of 61
The read sequence is CTTTCATTCTCTATCTTATATTACGGCACACACACA is at the index 19273894 to 19273930 in the genome sequence with a matching score of 64
The read sequence is CTTTCATTCTCTATCTTATATTACGGCACACACACA is at the index 19571697 to 19571733 in the genome sequence with a matching score of 56
The read sequence is CTTTCATTCTCTATCTTATATTACGGCACACACACA is at the index 20122192 to 20122228 in the genome sequence with a matching score of 61
The read sequence is CTTTCATTCTCTATCTTATATTACGGCACACACACA is at the index 20832327 to 20832363 in the genome sequence with a matching score of 64
The read sequence is CTTTCATTCTCTATCTTATATTACGGCACACACACA is at the index 22927341 to 22927377 in the genome sequence with a matching score of 69
The read sequence is GATTGCCTCTCATTGTCTCACCCTATTATGGGAAC is at the index 37 to 73 in the genome sequence with a matching score of 89
The read sequence is GATTGCCTCTCATTGTCTCACCCTATTATGGGAAC is at the index 495 to 531 in the genome sequence with a matching score of 89
The read sequence is GATTGCCTCTCATTGTCTCACCCTATTATGGGAAC is at the index 953 to 989 in the genome sequence with a matching score of 89
The read sequence is GATTGCCTCTCATTGTCTCACCCTATTATGGGAAC is at the index 1411 to 1447 in the genome sequence with a matching score of 89
The read sequence is GATTGCCTCTCATTGTCTCACCCTATTATGGGAAC is at the index 1869 to 1905 in the genome sequence with a matching score of 89
```

```
The read sequence is GTTGCCGCTAATCAAAAATAAATTCCTTGCAACATA is at the index 15801586 to 15801622 in the genome sequence with a matching score of 67
The read sequence is GTTGCCGCTAATCAAAAATAAATTCCTTGCAACATA is at the index 17468899 to 17468935 in the genome sequence with a matching score of 69
The read sequence is GTTGCCGCTAATCAAAAATAAATTCCTTGCAACATA is at the index 18294597 to 18294633 in the genome sequence with a matching score of 69
The read sequence is GTTGCCGCTAATCAAAAATAAATTCCTTGCAACATA is at the index 18332022 to 18332058 in the genome sequence with a matching score of 72
The read sequence is GTTGCCGCTAATCAAAAATAAATTCCTTGCAACATA is at the index 19089517 to 19089553 in the genome sequence with a matching score of 67
The read sequence is GTTGCCGCTAATCAAAAATAAATTCCTTGCAACATA is at the index 22073974 to 22074010 in the genome sequence with a matching score of 75
The read sequence is GTTGCCGCTAATCAAAAATAAATTCCTTGCAACATA is at the index 23055842 to 23055878 in the genome sequence with a matching score of 67
The read sequence is AAGCACAATAAGCCGCTCAAAAAAGGCATGAATA is at the index 5436 to 5472 in the genome sequence with a matching score of 100
The read sequence is AAGCACAATAAGCCGCTCAAAAAAGGCATGAATA is at the index 587700 to 587736 in the genome sequence with a matching score of 67
The read sequence is AAGCACAATAAGCCGCTCAAAAAAGGCATGAATA is at the index 1414634 to 1414670 in the genome sequence with a matching score of 69
The read sequence is AAGCACAATAAGCCGCTCAAAAAAGGCATGAATA is at the index 1645468 to 1645504 in the genome sequence with a matching score of 69
The read sequence is AAGCACAATAAGCCGCTCAAAAAAGGCATGAATA is at the index 1658201 to 1658237 in the genome sequence with a matching score of 67
```

Figure:3.4.2: Sample output of the final mapping(full output could not be screenshotted)

4. CONCLUSION:

There are various algorithms which can be used to create a Read Mapping software, but I had opted for the Seed and Extend Algorithm because it is very straightforward and is faster in execution. The execution time of finding the K-Mers for the complete genome sequence and mapping it with the reads for the complete reduced data set took only about 143 seconds in total. There can also be various Read Mapping algorithm which can be faster than this and provide a much more better execution time. But the mapping percentage of all the reads to the genome sequence was also perfect and matching score is also provided with the output. The execution time can be found below.



The screenshot displays a Jupyter Notebook interface. The top section shows four lines of text, each representing a read mapping result: 'The read sequence is TGTAGAGATTTTCGATATGAACGCATCGGCTTACCG is from index 19392481 to 19392517 in the Genome sequence with a matching score of 72', 'The read sequence is TGTAGAGATTTTCGATATGAACGCATCGGCTTACCG is from index 19427212 to 19427248 in the Genome sequence with a matching score of 67', 'The read sequence is TGTAGAGATTTTCGATATGAACGCATCGGCTTACCG is from index 28355844 to 28355880 in the Genome sequence with a matching score of 72', and 'The read sequence is TGTAGAGATTTTCGATATGAACGCATCGGCTTACCG is from index 28756315 to 28756351 in the Genome sequence with a matching score of 69'. Below this, a status bar indicates '1m 55s completed at 10:30 PM'. The bottom section shows a code cell with the command '[6]: genome_table = kmers(genome,10)' and a status bar indicating '28s completed at 10:27 PM'.

Figure:4.1: Execution time for the complete read mapping algorithm using seed and extend.

5. REFERENCES:

1. Lecture Notes and Lecture Videos
2. http://claresloggett.github.io/python_workshops/kmer_counting.html.
3. <https://sourmash.readthedocs.io/en/latest/kmers-and-minhash.html>
4. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3375638/> Mapping Reads on a Genomic Sequence: An Algorithmic Overview and a Practical Comparative Analysis
5. <https://stackoverflow.com/questions/32055817/python-fuzzy-matching-fuzzywuzzy-keep-only-best-match>

SUBNOTE: I was not able to attach the complete outputs because of the number of pages constraint for submitting the assignment and also was not able to capture the full screenshot.