CSE 549: Computational Biology, Fall 2017
Charu Mehndiratta (111493188) | Ritu Prajapati (111485620) | Saish Sali (111492587)

Project 5 Midway Report

Introduction

This report gives a brief overview of our implementation of MinHash and Containment Hash algorithms for long-read assembly.

Input:

- Two input files of sequencing reads
- K-mer size (k)
- Number of hash functions (h)

Output:

- Jaccard similarity between two sequencing reads using True Jaccard, MinHash and Containment hash algorithm for all vs all alignment required to determine overlapping read pairs.

Conventional Approach - True Jaccard

True Jaccard refers to a conventional method of calculating similarity between two strings. The algorithm we implemented for sequencing reads is as follows:

1. Take two sequencing reads.
2. Divide both the reads into k-mers and populate them into corresponding sets.
3. Find intersection of the two sets.
4. Find union of the two sets.
5. Calculate the jaccard index by dividing the intersection of the sets by the union of the sets.

Issues with conventional approach:
Total number of elements in each set depends on the length of the read and as the length of the read increases the number of elements increases. In the worst case the complexity of finding jaccard similarity can be $O(mn)$ (where m is the length of the first sequencing read and n is the length of the second sequencing read). This problem will be solved by MinHash and Containment Hash (as described below) where the size of the set is bounded by the number of hashes.

Min hash Approach

In conventional genome assembly methods, the all-versus-all alignment is required to determine the overlapping read pairs. This overlapping step itself incur a significant computational cost. So a new approach named MinHash is used to represent the sequencing reads in a more compact version.

The basic idea of minimum hash algorithm is to create a hash signature for the input reads and compare the signatures to compute the overlap. As the sketch size (collection of min-mers) is significantly smaller than the total number of k-mers to be compared, the overall performance improves due to reduced overlapping calculation.

Algorithm:
1. Take two sequencing reads as input.
2. Generate a random hash vector upto the size of sketch.

3. Select a sequence one by one and generate k-mer substrings.
4. For each k-mer, generate all the hash functions required in the sketch and keep only the one that has the smallest value.
5. Together with sketch values, keep the k-mer that generated the min-hash value for ith hash function. This k-mer string will later be used in the sequencing procedure.
6. For calculating the hash functions, we used MurmurHash3. Murmur hash is a non-cryptographic function which uses three operations multiply, rotate and XOR to provide the hash value.
7. As now we have the sketch for both the reads, the jaccard similarity can be calculated using the intersection and union on these two sketches.
8. The jaccard similarity between two sequencing reads is calculated as:
   Number of similar hash function in the sketch of two reads / length of the sketch
   Greater the value of jaccard similarity (near 1), greater the similarity between two sequencing reads.

Issues with MinHash approach:
- The performance of minHash approach reduces when the two sequencing reads are different in size.
- The performance is dependent on kmer size and hash functions used. So, an optimal value for k and number of hash functions is required to obtain accurate results.

Containment Hash Approach
- Suitable for estimating the Jaccard index of sets of very different size
- Application: Detect the presence or absence of a given genome in a metagenomic sample
- Significantly more accurate
- Smaller computational complexity
- Utilizes less memory than the traditional min hash approach

Approach:
1. Randomly sample elements only from the smaller set (say, set A)
2. Use bloom filter probabilistic technique to quickly test if this element is in set B (and hence in $A \cap B$)
3. This is used to estimate the containment index, which is then used to estimate the Jaccard index itself

Algorithm:
1. For the larger sequencing read (say, s_1), instantiate and configure bloom filter B with projected number of elements (l_1 - k + 1), false positive probability (p) of 0.001 and random seed
2. Add kmers for sequencing read s1 to the bloom filter
3. Generate shingles for sequencing read s_2
1. Convert all k mers generated in step 3 to integer fingerprints using multiple, randomized hash functions (say, Murmur3 hash with random seed values)
2. For each hash function, only the shingle corresponding to the minimum valued fingerprint, or min-mers is retained. The collection of min-mers of a sequence makes the sketch.
3. For all the kmers in the sketch of sequencing read s_2, query for its existence in the bloom filter B. Compute number of such memberships in Y.
4. Compute containment estimate, C = (Y / h) - p
5. Compute Jaccard index = $(C * l_2) / (l_1 + l_2 - l_2 * C)$

Instructions to execute program:

```
# make
# ./sequence_similarity
```

Results

Sequencing read 1:
CATGGTTATTATTACATGGTTATTATTACACATGGACCCATGGACCGAGGACCATGGACCGGTAGACA
TGGACCGGTAGACATGCCTAGACCAGCATGCCTAGACCAGTTTAACCCTTAACTCGGACCCTCTATTA
CACATG

Sequencing read 2:
CATGGTTATTATTACATGGTTATTATTACACATGGACCCATGGACCGAGGACCATGGACCGGTAGACA
TGGACCGGTAGACATGCCTAGACCAGCATGCCTAGACCAGTTTAACCCTTAACTCGGACCCTCTATTA
CACATGGACCCATGGACCGAGGACCATGGACCGGTAGACATGGACCGGTAGACATGCCTAGACCAGC
ATGCCTAGACCAGTTTCATTTATTATTACCCTTAACTCGGACCCTCATTTACCCGTAAAAAGGGGGGGA
TGGACCGGTAGACATGGACCGGTAGACATGCCTAGACCAGCATGCCTAGACCAGTTTGATGGTTATTA
TTACATGGTTATTATTACACATGGACCCATGGACCGAGGACCATGGACCGGCCTAGACCAGTTTCCGG
TAGACATGCCTAGACCAGCATGCCTAGACCCCATGGTTATTATTACATGGTTATTATTACACATGGACC
CATGGAACCCTTAACTCGGACCCTCATTTACCCGTAAAAAGGGGGGGATGGACCGGTAGACATGGAC
CGCCATGGACCGGTAGACATGGACCGGTAGACATGCCTAGACCAGCATGCCTAGACCAGTTTCATGGT
TATTATTACATGGTTATTATTACACATGGACCCATGGACCGAGGACC

Number of hash functions: 200

K mer size: 1 to 15

False positivity: 0.001

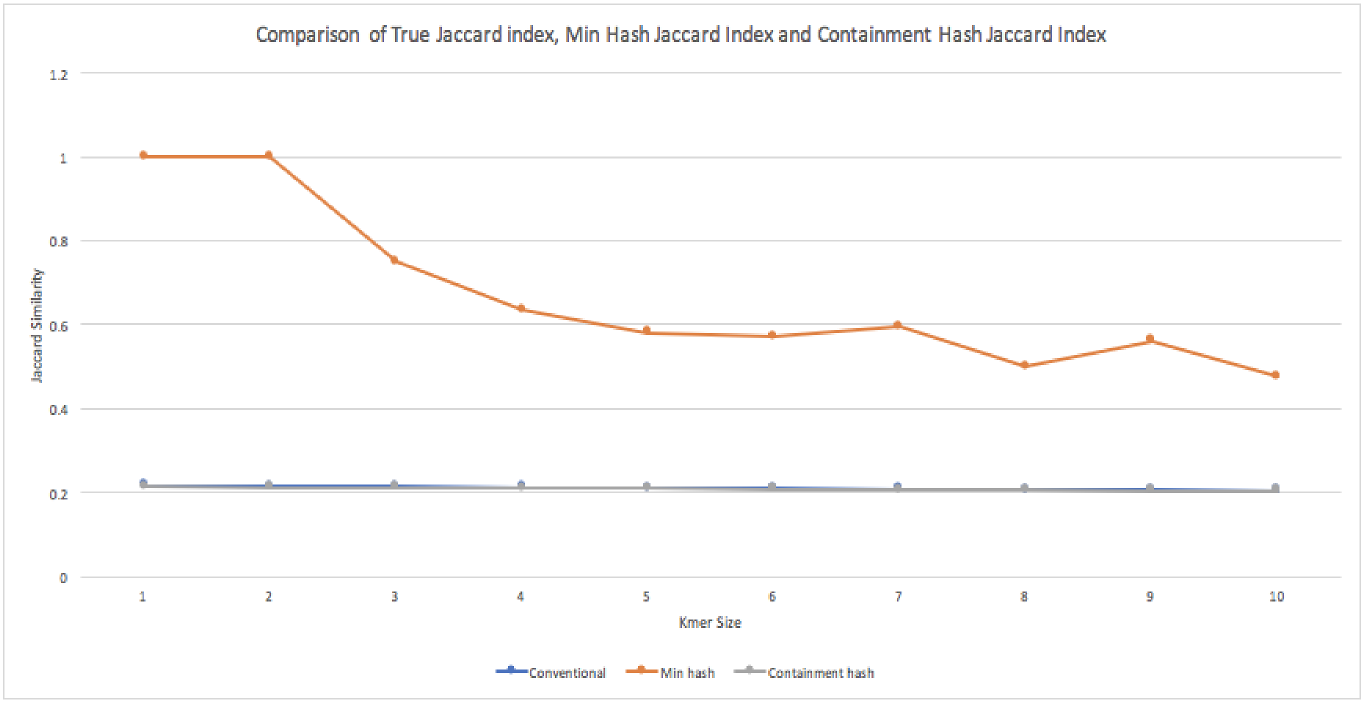

Figure 1: Comparison of True Jaccard Index, Min hash Jaccard Index and Containment Hash Jaccard Index for two sequencing reads

Kmer Size	True Jaccard Index	Min Hash Jaccard Index	Containment Hash Jaccard index
1	0.215478	1	0.21417
2	0.214286	1	0.212986
3	0.21309	0.75	0.211799
4	0.21189	0.635	0.210608
5	0.210687	0.58	0.209413
6	0.20948	0.57	0.208215
7	0.20827	0.595	0.207013
8	0.207055	0.5	0.205807
9	0.205837	0.56	0.204597
10	0.204615	0.475	0.203384
11	0.20339	0.465	0.202167
12	0.20216	0.46	0.200947
13	0.200927	0.485	0.199722
14	0.19969	0.425	0.198494
15	0.19845	0.48	0.197262

Table 1: Computation of True Jaccard Index, Min hash Jaccard Index and Containment Hash Jaccard Index for different kmer sizes on two sequencing reads

References -

- https://github.com/dkoslicki/MinHashMetagenomics
- https://www.biorxiv.org/content/biorxiv/early/2017/09/04/184150.full.pdf
- http://www.nature.com.proxy.library.stonybrook.edu/articles/nbt.3238.pdf
- http://mccormickml.com/2015/06/12/minhash-tutorial-with-python-code