Project Midway Report

Project: Long Read Mapping Algorithms

Deliverables:

- Implemented Min Hash approach
- Implemented Containment min Hash approach
- Compared both approaches with simulated data. Plotted graphs for comparison.

Algorithm:

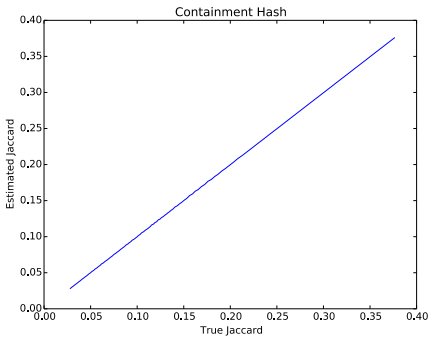
- ✓ **Min Hash:**

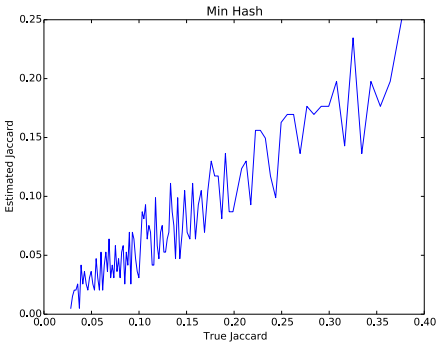
Step 1: Generate kmers for string A and B
Step 2: Generate number of hash functions
Step 3: Get Hashes. For each hash function, find min hashes and the kmers that give the min hashes
Step 4: Repeat step 2 and step 3 for string B
Step 5: Go through the min hashes of A and B, find intersection with common elements and find Union (which is size of A + size of B – intersection)
Step 6: Calculate Estimated Jaccard Index as: (intersection / A U B)

- ✓ **Containment Hash:**

Step 1: Generate kmers of A and B
Step 2: Populate bloom filter with kmers of larger set B
Step 3: Get the kmers which give min hash values for the hash functions
Step 4: Calculate intersection by checking how many of the min hash kmers of A from Step 3 are in the bloom filter
Step 5: Adjust for false positive rate
Step 6: Estimate containment index
Containment Index = intersection / (float)hash_count
Step 7: Estimate Jaccard index from Containment Index
Estimated Jaccard Index =
(size_A * containment_index) / (size_A + size_B_est - size_A * containment_index)

Results:





Jaccard estimate from Containment Hash is closer to the true jaccard index compared to the estimate from Min Hash. It is also faster to find jaccard estimate through containment min hash approach.

```
hash count    = 100
false pos rate=0.001
A size = 142
B size = 659
kmer size | est jaccard (min) | (containment) | true jaccard
 ---> 128 |    0.005025 |     0.028166 |    0.028195
 ---> 127 |    0.015228 |     0.030044 |    0.030019
 ---> 126 |    0.020408 |     0.031802 |    0.031835
 ---> 125 |    0.020408 |     0.033736 |    0.033645
 ---> 124 |    0.025641 |     0.035411 |    0.035448
 ---> 123 |    0.005025 |     0.037275 |    0.037244
 ---> 122 |    0.041667 |     0.038993 |    0.039033
 ---> 121 |    0.025641 |     0.041002 |    0.040816
 ---> 120 |    0.036269 |     0.042548 |    0.042593
 ---> 119 |    0.025641 |     0.044480 |    0.044362
 ---> 118 |    0.020408 |     0.046077 |    0.046125
 ---> 117 |    0.030928 |     0.048098 |    0.047882
 ---> 116 |    0.036269 |     0.049580 |    0.049632
 ---> 115 |    0.025641 |     0.051702 |    0.051376
 ---> 114 |    0.020408 |     0.053058 |    0.053114
 ---> 113 |    0.047120 |     0.055190 |    0.054845
 ---> 112 |    0.030928 |     0.056510 |    0.056569
 ---> 111 |    0.020408 |     0.058439 |    0.058288
 ---> 110 |    0.052632 |     0.059936 |    0.060000
 ---> 109 |    0.020408 |     0.062205 |    0.061706
 ---> 108 |    0.041667 |     0.063338 |    0.063406
 ---> 107 |    0.052632 |     0.065266 |    0.065099
 ---> 106 |    0.036269 |     0.066836 |    0.066787
```

Limitations:

Most accurate results when kmer sizes are > 15
Graphs are not as smooth as those in the original paper

Resources:

- Paper: https://www.biorxiv.org/content/biorxiv/early/2017/09/04/184150.full.pdf
- https://github.com/dkoslicki/MinHashMetagenomics
- https://www.geeksforgeeks.org/vector-in-cpp-stl/
- http://infolab.stanford.edu/~ullman/mmds/ch3.pdf
- https://matthewcasperson.blogspot.com/2013/11/minhash-for-dummies.html
- https://stackoverflow.com/questions/5008804/generating-random-integer-from-a-range

- Hash Functions generation page 188 in https://www.eecs.harvard.edu/~michaelm/postscripts/rsa2008.pdf
- Bloom filter http://www.partow.net/programming/bloomfilter/index.html
- Bloom filter https://archive.codeplex.com/?p=libbloom and https://github.com/ArashPartow/bloom
- Murmur Hash https://github.com/aappleby/smhasher
- https://stackoverflow.com/questions/20052674/how-to-convert-vector-to-set
- https://stackoverflow.com/questions/11989374/floating-point-format-for-stdostream
- Plotting graph https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/
- https://www.uow.edu.au/~lukes/TEXTBOOK/notes-cpp/