INTRODUCTION TO FOCUS AREAS IN BIOINFORMATICS - WS19/20

Project 13: Comparing two genomes based on overlapping k-mers

Raghavendra Tikare and Stanislav Klein

Full list of author information is available at the end of the article

Abstract

Goal of the project: This week's project was centered on comparing two similar genome sequences, assuming they have many overlapping k-mers.

Main result(s) of the project: Comparing the X-chromosomes of humans and mice with a Burrows-Wheeler-Transformation (BWT) approach and a Bloom Filter approach resulted in differences regarding runtime.

Personal key learnings: There are different algorithms on how to analyze genomic data which differ in runtime. Understanding the concepts and choosing and fitting algorithm is important to solve a task in an efficient way.

Estimation of the time: 16 hours

Project evaluation on a scale of 1-5: 2

Word count: 551

Keywords: genome comparison; k-mers; mouse; human

Description of the data

Our main goal was to compare two similar genome sequences, so we decided to go with the X-chromosome of a human and X-chromosomes of a mouse. We found their respective genome sequences on the National Center for Biotechnology Information website, which can be found on https://www.ncbi.nlm.nih.gov/guide/howto/dwn-genome/. They are quite similar in size (Humans' 44 MB as opposed to mice' 45 MB compressed data), we thought it would be interesting to see how similar sex chromosomes of two seemingly completely different organisms actually are.

Implementation

Burrows wheeler transform(BWT) based Search

The implementation of the BWT based approach was highly influenced from the results of the previous week's project on index-based and index-less search, since we already had the foundations and additional assistance from Gunjureddy et.al. Most of the code stayed the same, except for obvious sections, e.g. changing input files.

Bloom Filter based Search

For this approach we did some research and came across code by Arash Partow on Github found on http://www.partow.net/programming/bloomfilter/index.html. In his article he demonstrates a simple bloom filter and also provided a free to use bloom filter library containing a header file and additional documentation. Given the basic structure we then needed to modify the code in a way that it reads our data.

Benchmarks

In addition to implementing the BWT-based and the bloom filter approach we had to implement the code and output a timer to get the benchmarks as seen in the figure 1 and 2 respectively. Here we obtain quite similar run times when considered with the X chromosomes of human and mouse genome. It took nearly 4 minutes to build the index for BWT and less than a minute for a bloom filter with the number of hash functions used and bit filter size are given in the table in the figure 1 and 2 respectively.

Algorithm	False positive rate in %	Wall clock time in seconds (s)	Number of hash functions used	Filter size in bits
BWT	---	37.896	---	---
Bloom filter	0.1	0.003	12	4658156
Bloom filter	0.001	1.250	18	7584215
Bloom filter	0.00001	2.336	25	9102584

Figure 1 Benchmark results of k-mer size of 10

Algorithm	False positive rate in %	Wall clock time in seconds (s)	Number of hash functions used	Filter size in bits
BWT	---	46.389	---	---
Bloom filter	0.1	0.227	12	4658156
Bloom filter	0.001	1.584	18	7584215
Bloom filter	0.00001	3.558	25	9102584

Figure 2 Benchmark results of k-mer size of 15

Discussion

As seen in the previous figures upon analyzing the data, the wall clock time of both the BWT algorithm and the bloom filter algorithm are very different. The bloom filter algorithm in particular shows a much faster run time than the BWT algorithm. That is because it uses hash functions to determine a possible match, but it makes it less robust because of being a probabilistic approach as it also has a false positive rate. This also matches the time building the index in both algorithms as described in the benchmark section. Depending on how we set the allowed false positive rate, the run time of the bloom filter algorithm increases. Despite taking a long time to build the index, the BWT algorithm seems to be more precise when it comes to analyzing data.