Basic Contamination (BACON) Search Algorithm

COMPUTATIONAL GENOMICS

1. ABSTRACT

During next generation sequencing, researchers may encounter DNA reads that have been contaminated by various foreign substances. Sources of contamination can include bacteria from the surrounding environment, human skin cells from the researcher handling the sample, or cross-contaminated petri dishes of two different species' DNA. We approached the problem of DNA sequence contamination with the goal of determining if a DNA sample is contaminated and identifying the source of contamination if it exists. Although there are existing algorithms to identify contamination, our aim was to create a novel simple search algorithm that effectively analyzes unknown contamination. Our resulting algorithm has 72.6% error from what we intended, therefore it has room for improvement.

2. INTRODUCTION

Contamination in genome sequencing is a major issue in genomic research that can have far reaching implications for the entire field. One incorrect conclusion based on contaminated data can result in a cascading effect which can cause entire research papers to be rescinded due to inaccurate premises. Even with the improvement in sequencing technology and the specific countermeasures that researchers take, contamination in sequencing reads still occurs often. One major reason is that contamination can come from a variety of sources, ranging from cross-individual to cross-species to within-individual.

Currently, there exist several methods to address this issue. However, generally, most of the methods require comparison against an existing data set of known contaminants and therefore cannot identify unknown contaminations. Other methods require specific software-generated inputs, environmental (metagenomic) related data, or additional external data in combination with DNA sequencing reads, instead of inputting simple read files in FASTA or FASTQ format. In this report, we theorize and implement a method that will allow us to determine basic unknown contaminations from a set of reads. The only assumption that our algorithm makes is that the amount of contamination is a relatively small amount. We are most interested in situations where the researcher follows correct laboratory guidelines, so generally this our assumption should be true.

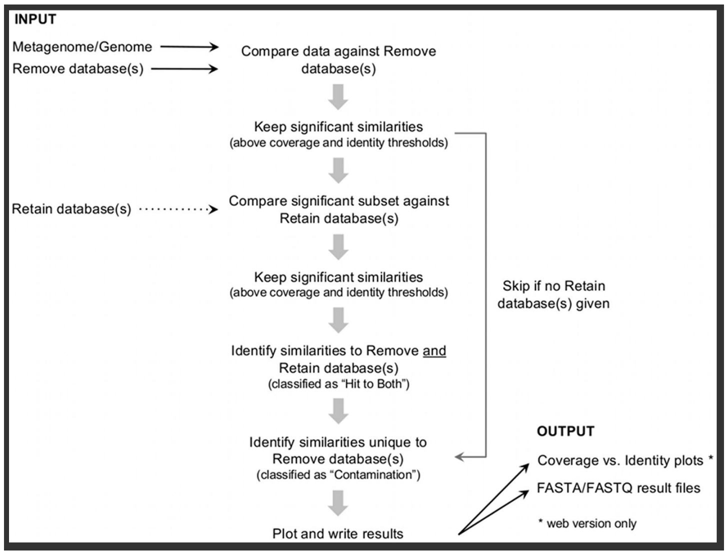
There exist several major issues with identifying unknown contaminants from a set of reads. First, we do not know what the contaminants will look like and what the actual ideal, uncontaminated genome looks like. This means we cannot incorporate any information about the genomes in our model. Second, there is often no prior-known information about the location of these contaminants in the reads, or about the composition of these contaminants. So, all of the aforementioned information must be collected by analyzing the set of reads. Finally, runtime is an issue with any genomic algorithm due to extremely large datasets.

Once the ability to identify unknown contaminants is established, there are many other pathways that can be explored. With better detection abilities, pathogens and viruses can be identified in sequences. This would allow for better diagnosis of human diseases that are gene-based.

3. PRIOR WORK

"Fast Identification and Removal of Sequence Contamination from Genomic and Metagenomic Datasets"

In 2010, a duo from San Diego State University, developed a tool, DeconSeq, which detects and removes contamination in longer read samples. The goal of the research was to identify and remove human contamination from metagenomic samples, however the tool is able to be applied to any sequence contamination. The DeconSeq tool sorts contamination sequences, and "eliminates redundant hits with higher similarity to non-contaminant genomes..." The DeconSeq was accurate 99% for each simulated metagenome sample. However, it is notable that these results were based on thresholds derived from previous work. Below is the workflow of the DeconSeq software.


INPUT
Metagenome/Genome
Remove database(s)
Compare data against Remove database(s)
Keep significant similarities (above coverage and identity thresholds)
Retain database(s)
Compare significant subset against Retain database(s)
Keep significant similarities (above coverage and identity thresholds)
Skip if no Retain database(s) given
Identify similarities to Remove and Retain database(s) (classified as "Hit to Both")
Identify similarities unique to Remove database(s) (classified as "Contamination")
Plot and write results
OUTPUT
Coverage vs. Identity plots *
FASTA/FASTQ result files
* web version only

"ContEst: estimating cross-contamination of human samples in next-generation sequencing data"

In 2011, a group from Broad Institute of MIT and Harvard, a biomedical and genomic research center, published a paper about a new software tool they developed which estimates level of cross-contamination in sequencing data. The tool, ContEst, given sequencing data and a genotyping array, calculates the "maximum *a posteriori* estimate" (estimate of an unknown quality) of contamination in sequencing data based on the base identities and quality scores. It was tested on data with known contaminants of varying levels, to examine the software's accuracy and then was focused towards low-level cross-individual DNA contamination. In Silico datasets were created by mixing known contaminants of varying but specific levels with a

"primary sample" containing a contamination level of 0.08%. The ContEst tool was able to accurately report the percentage of contamination in each trial with varying conditions.
"Detecting and Estimating Contamination of Human DNA Samples in Sequencing and Array-Based Genotype Data"
In 2012, by a joint effort from the Department of Biostatistics and Center for Statistical Genetics at the University of Michigan and the Center for Inherited Disease Research at The Johns Hopkins University, methods for detecting within-species contamination were developed based on sequencing reads and array based genotype data, sequencing reads alone, and array based genotype data alone. Analysis of sequencing reads allows for the detection of contamination after the sequencing data is generated, while the array-based genotype data allows for detection prior to sequencing generation. The methods were tested using in silico contaminated and intentionally contaminated samples and detection of contamination was accurate for levels as small as 1%. Software developed from the methods include: *VerifyIDintensity* which detects and estimates contamination using "intensity' data (sequencing reads *and* array based genotype data); *VerifyBamID* which verifies whether data being generated matches expectations, and identifies if the reads are contaminated as a mixture of the two samples; *BAFRegress* which detects and estimates contamination from genotyping arrays with a regression model.

"Conpair: concordance and contamination estimator for matched tumor-normal pairs"
In 2016, at the New York Genome Center, a goal to develop sequence-only based methods led to the creation of a tool, Conpair, to detect contamination. While VerifyBamID and ContEst are the standard methods for detecting sample contamination, Conpair detects contamination in cancer studies based on sequencing data alone. The method detects contamination levels as low as 0.1% which is a higher accuracy than ContEst and VerifyBamID. In addition to being a faster method, it does not require additional input data like the other two methods. The Conpair method takes in a reference genome and "a short list of pre-selected highly informative genomic markers that are provided" and runs a concordance verification and contamination estimation.

The BACON search algorithm aims to analyze similar contamination information as the previously mentioned methods, but its implementation was not based on the previously mentioned tools.
Beyond the scope of our specific area of interest, we found there exists DNA contamination research with different goals. For example, there are studies that aim to find specific bacteria in the genome, studies that focus more on cross-species contamination or cross-individual, and studies that apply contamination algorithms to different datasets such and RNA sequence, or expressed sequence tags (ESTs) for predicting new genomes and splicing new proteins. In addition to the relatively new computational research applied to the genome, there is research conducted in a purely biological and chemical context.

4. METHODS AND SOFTWARE

The initial step of the project was to decide on the type of detection system to research. There are multiple possibilities, such as determining the species of bacteria that exist in a sample of soil to finding the amount of mycoplasma contamination in genome sequencing. The major distinction between these different possibilities is whether to search for known sequences, such as finding k-mers that belong to distinct species, or to search for unknown sequences, such as determining which sequences do not belong in the overall genome.

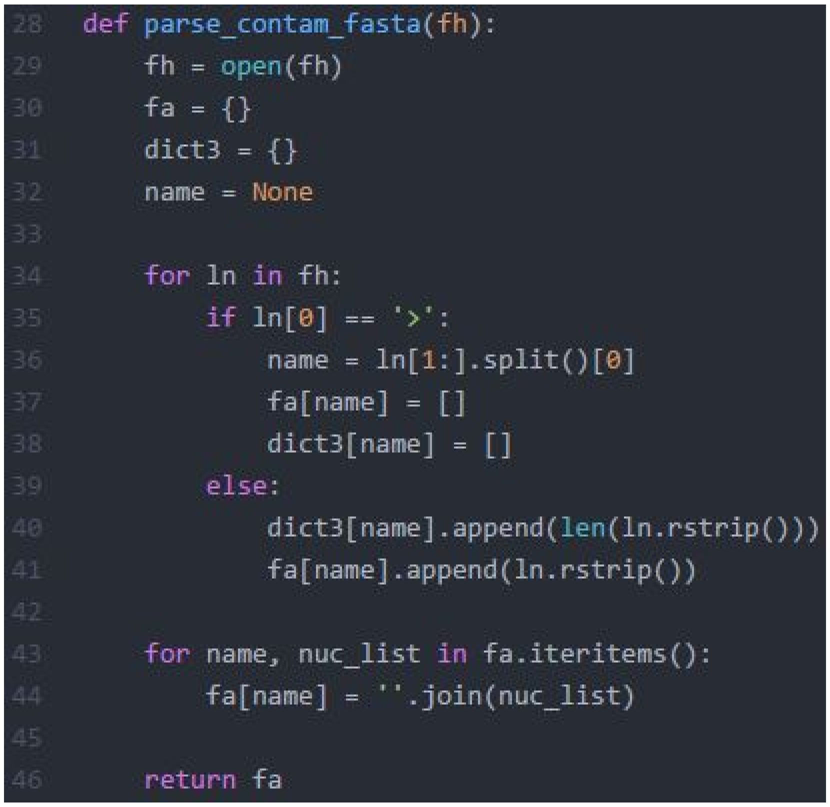
The chosen goal was to search for unknown contaminants in a sample of reads. In order to complete this objective, we had several main objectives. (1) Write the software that will analyze the data and determine contaminants. (2) Generate sample data that contains contaminations (of various amounts) in order to test the results. (3) Analyze the results of our software given specific parameters determining the sample data.

4.1 DETECTION SOFTWARE

The BACON software was split into four main sections. (1) Reading and manipulating the input from a FASTA file. (2) Analyzing the similarity between different reads. (3) Analyzing the occurrence of certain sequences. (4) Interpreting the output of (2) and (3) to display contamination amount and bad reads.

4.1.1 Input

The entire software takes a FASTA file as the input. Inside the FASTA file is a list of reads that represents an attempt at sequencing part of a genome. However, unknown contaminants may appear. The reads are parsed and stored in a dictionary, with the id and sequence as the key and value. The dictionary is then passed to the two algorithms that will analyze the data for contaminations.



```python
def parse_contam_fasta(fh):
    fh = open(fh)
    fa = {}
    dict3 = {}
    name = None

    for ln in fh:
        if ln[0] == '>':
            name = ln[1:].split()[0]
            fa[name] = []
            dict3[name] = []
        else:
            dict3[name].append(len(ln.rstrip()))
            fa[name].append(ln.rstrip())

    for name, nuc_list in fa.iteritems():
        fa[name] = ''.join(nuc_list)

    return fa
```

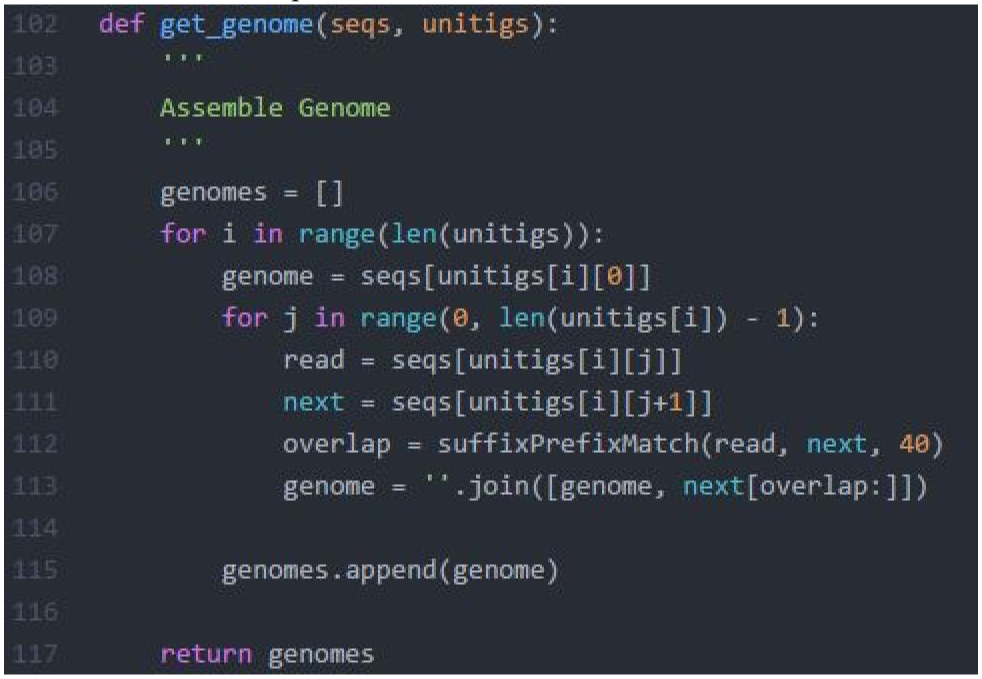
4.1.2 Sequence Similarity

The initial method for finding the similarity between reads used an overlap graph algorithm in order to determine how much overlap a read had with other sequences in the data. The assumption that this method is based on is that if a read's sequence has very little overlap with other reads', then it has very little similarity. Distinct sequences indicate that the read is possibly a contamination, as contamination sequences should not overlap with actual genome sequences. The overlap graph algorithm uses a method similar to that discussed in class. For every read A, we compare it to every other read for front and back in order to get the max suffix and prefix of A shared with some other read. We then know how much of each read is overlapped. The total overlap value determines whether or not a read is a contaminant.

However, after doing some consideration, several issues with this method appeared. The major problem was that if part of a contamination genome occurs multiple times, contaminated reads will have high overlap values and be missed when searching for threshold reads. This can occur if there are a high number of contaminated reads or if the contaminant genomes are very short.

In order to address this problem, we built unitigs from the overlap graphs. Once the unitigs were build, the resulting sequences should represent either the actual genome or contaminants. The theory behind this method is that the longest unitig should represent wanted genome, while shorter unitigs should represent contaminants. However, using just the unitigs can be misleading. This is because the actual genome may be made up of more than one unitig, which may lead to the actual genome unitigs being shorter than contaminant unitigs. In this case, the incorrect reads will be registered as contaminants.
After these considerations, a final modified implementation of the unitig method was created. In this algorithm, unitigs were created from the overlap graph. This allows us to group reads by similarity. An overlap graph was then built from the unitigs themselves. Any overlap between

unitigs over a threshold value was an edge on the graph. After every possible overlap was considered, the result is a graph that has islands representing the actual genomes and contaminants. This is because contaminants should have no overlap with the actual genome, so the unitigs should have no overlap as well.



```
def get_genome(seqs, unitigs):
    '''
    Assemble Genome
    '''
    genomes = []
    for i in range(len(unitigs)):
        genome = seqs[unitigs[i][0]]
        for j in range(0, len(unitigs[i]) - 1):
            read = seqs[unitigs[i][j]]
            next = seqs[unitigs[i][j+1]]
            overlap = suffixPrefixMatch(read, next, 40)
            genome = ''.join([genome, next[overlap:]])

        genomes.append(genome)

    return genomes
```

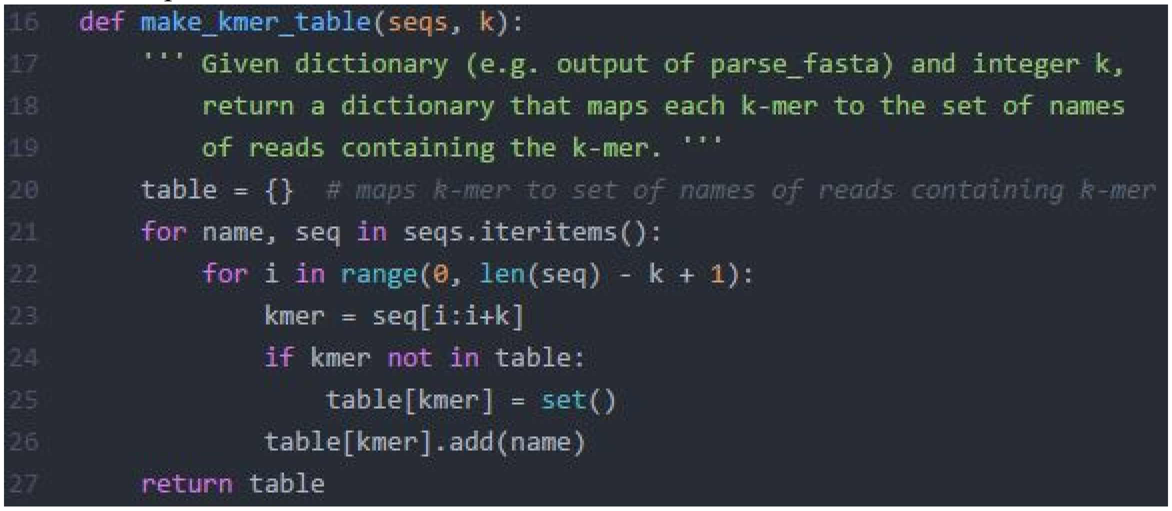
There is still a remaining issue of deciding which is the genome and which is the contaminant. An educated guess would suggest that the bigger island of unitigs will be the the actual genome while the smaller islands will be contaminants.

4.1.3 Sequence Occurrence

The initial algorithm for finding the number of times a sequence occurs was using local alignment to compare every read to every other read. If the alignment score (hamming distance) was less than a threshold value, determined by the length of the read sequences, then the two reads were associated with each other.

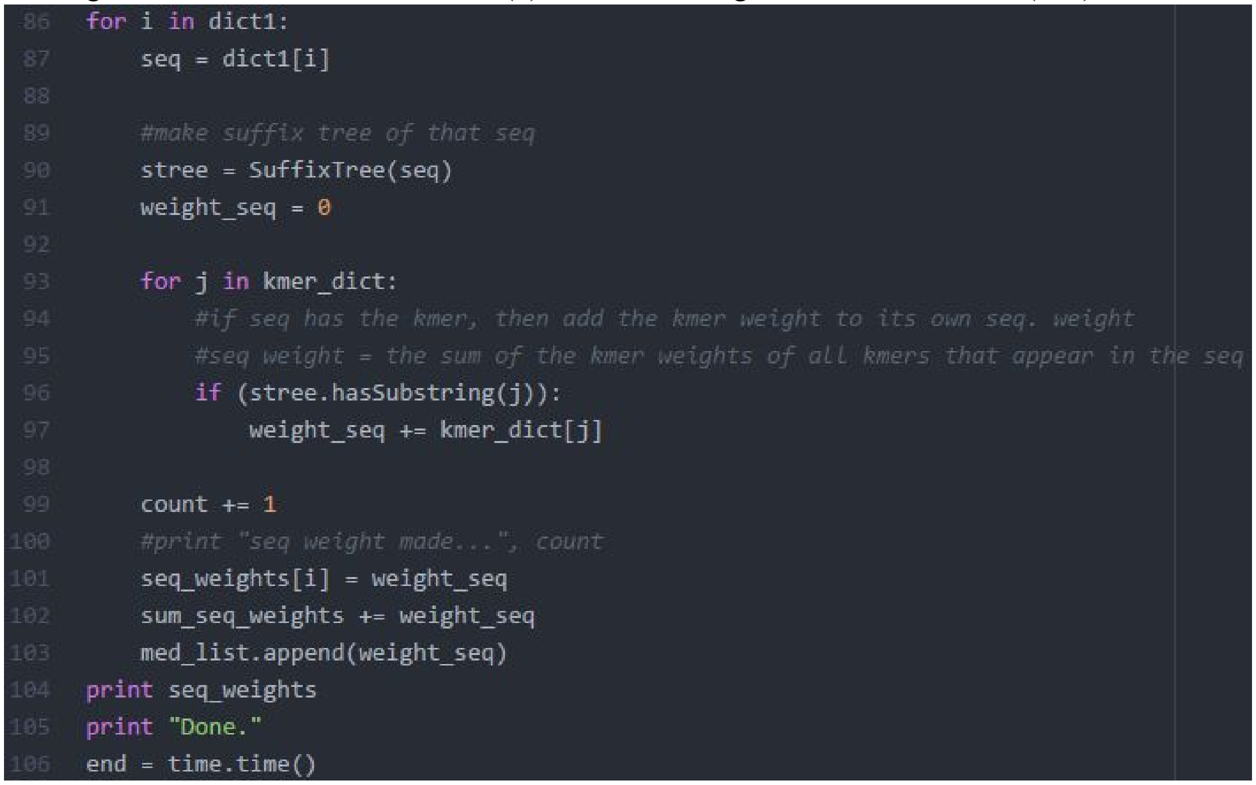
However, after writing and testing the algorithm for this algorithm, several issues were determined. First, the amount of time required is absurd. The time complexity is O(n^2), where n is the number of reads. One test case had 5000 reads. Doing local alignment for one read against every other of the 5000 read took over 3 minutes, resulting in a total time of 10 days if the program was fully run. Second, the number of times a section of the contaminant occurs is highly dependent on the size of the contaminant and the total contamination percentage. Similar to sequence similarity, if the contaminant is short, then sections of the contaminant will have high occurrence. This will prevent them from being caught when the reads with occurrence rates less than the threshold are analyzed.

In order to solve the speed issue, we use new implementation which will henceforth be referred to as the read kmer weight algorithm. What this algorithm does is go through every single read in the data set and create kmers from the read, where k is determined by sequence total length. These kmers are added to a dictionary where their occurrence rate is recorded as a ‘weight’. This will be the first pass.



```python
def make_kmer_table(seqs, k):
    ''' Given dictionary (e.g. output of parse_fasta) and integer k,
        return a dictionary that maps each k-mer to the set of names
        of reads containing the k-mer. '''
    table = {}  # maps k-mer to set of names of reads containing k-mer
    for name, seq in seqs.iteritems():
        for i in range(0, len(seq) - k + 1):
            kmer = seq[i:i+k]
            if kmer not in table:
                table[kmer] = set()
            table[kmer].add(name)
    return table
```

Then in a second pass through the reads, the total weight of all the kmers in the read will be summed and assigned to the read. The theoretically result is that contaminated reads will have lower kmer sum weights since their kmers should occur less often. This method increases the speed dramatically, with a time complexity of O(n*d), where n is the number of reads and d is the length of the read. If n >> d, then O(n), which is a magnitude smaller than O(n^2).



```python
for i in dict1:
    seq = dict1[i]

    #make suffix tree of that seq
    stree = SuffixTree(seq)
    weight_seq = 0

    for j in kmer_dict:
        #if seq has the kmer, then add the kmer weight to its own seq. weight
        #seq weight = the sum of the kmer weights of all kmers that appear in the seq
        if (stree.hasSubstring(j)):
            weight_seq += kmer_dict[j]

    count += 1
    #print "seq weight made...", count
    seq_weights[i] = weight_seq
    sum_seq_weights += weight_seq
    med_list.append(weight_seq)
print seq_weights
print "Done."
end = time.time()
```

One of the issues that remain is that the contamination rate cannot be high for this method. If the contamination rate is high, then the kmer weight of the kmers belonging to contaminated reads will be higher than that of the actual genome and cause incorrect results.

4.1.4 Output

The output will contain 3 main values. They are the percentage contamination according to the unitig creation algorithm (or the sequence similarity method), the percentage contamination according to the read kmer weight algorithm (or the sequence occurrence method), and the combined percentage contamination method. The combined percentage output is based on the lists of reads that are found in each algorithm. Only reads that occur in both are counted as "definite" contaminations for the combined percentage score.

4.2 CREATING SIMULATED TEST DATA

There were initially two ideas on how the create the simulated test data. First was create an example genome, a contamination genome, combine the two, and then create a FASTA read file. Second was create the target and contaminations genomes, create individual FASTA read files for each genome, and then combine the FASTA files into one. After some analysis, the first method does not allow for us to test contamination at all. It essentially just simulates one data set without contaminations. The second method is the correct method - each read corresponds to one possible source.

In order to create the simulated data, several methods were used. First, a random generator to create contamination was written in java. This program allowed us to change specific variables related the the contamination genome, such as length and nucleotide preference. Then, a program called metasim was used to simulate the actual FASTA file full of reads. Metasim allows for us to control many variables such as mutation rate and read length when generating the reads. It also has different modes for different types of genome sequencing.

4.3 RANGE OF TESTING

Using the method described above, a range of parameters can be described and tested. For our specific testing, we did a range of contamination length and number of reads.

5. RESULTS

We only used simulated data, where the list of contaminants and the percentages of contaminants in our k-mer and unitig data is displayed. Also, the total percentage of contaminants of both sections combined is displayed.

Total Percentage: 2.76 % , where the expected percentage is 10%. Our code underestimates the actual percentage of the amount of contaminants. We could not find any other similar code to compare to.

6. CONCLUSIONS

We implemented both a sequence weight algorithm and found all possible unitigs for the FASTA file input. Both algorithms were combined to estimate the total percentage of contaminants in any given FASTA file. The weakness in our code would be the fact that we could not assemble the genomes of the contaminated genome and the contaminant genome and calculate the percentage in that manner.

We had several stretch goals that were not meet during the time period given for the project. One of the stretch goals was converting the nucleotide sequences to amino acids via codons and determining whether or not this had a beneficial effect on the speed and accuracy of contamination detection. The overall algorithm should not change in order to implement this, however the different possible amino acid sequences based on different starting positions in the nucleotide sequence will cause a triple increase in comparisons. There are several possible benefits of switching to amino acids such as ignoring harmless mutations, and several possible detriments such as increased required time.

Another possible addition for coding is adding the ability to change operations via command line arguments and adding certain minor functionality. For example, having the ability to add the exact positions of all the contaminants is possible with the methodology used. However, this was not added for the time being due to some concern with time and space complexity. One of our initial goals was to make it as efficient and fast as possible, so we removed or didn’t add some possible options that would be useful in a practical application.

Since the application has the ability to identify reads that are marked as contaminated by our algorithm, once this functionality is fully added, it opens up a range of possibilities. The entire methodology can be adapted to just finding sections of the reads that are parts of different genomes. This will allow users to find things such as bacteria and viruses that are in samples, which will help in the identification of diseases. Additionally, bacteria and viruses cause many diseases, so by identifying these ‘contaminants’ in a data set, we can allow for early detection and prevention efforts.

7. LITERATURE CITED

ArXiv, E. T. (2012). Human Genome Contaminated With Mycoplasma DNA. Retrieved December 09, 2016, from https://www.technologyreview.com/s/424458/human-genome-contaminated-with-mycoplasma-dna

Bergmann, E. A., Chen, B., Arora, K., Vacic, V., & Zody, M. C. (2016, October 15). Conpair: Concordance and contamination estimator for matched tumor–normal pairs. *Bioinformatics*, 32(20): 3196–3198. doi: 10.1093/bioinformatics/btw389

Gibbon, V. E., Štrkalj, G., Harington, J., & Penny, C. B. (2008). A review of DNA analyses of archaeological and ancient tissues. *Transactions of the Royal Society of South Africa,* 63(2), 145-149. doi:10.1080/00359190809519218

Jun G., Flickinger M., Hetrick K.N., Romm J.M., Doheny K.F., Abecasis G.R., Boehnke M., Kang H.M. (2012). Detecting and Estimating Contamination of Human DNA Samples in Sequencing and Array-Based Genotype Data. *The American Journal of Human Genetics,* 91(5), 839-848. doi: 10.1016/j.ajhg.2012.09.004

Pearson, W. R. (2013). An Introduction to Sequence Similarity ("Homology") Searching. *Current Protocols in Bioinformatics.* 42:3.1:3.1.1–3.1.8. DOI: 10.1002/0471250953.bi0301s42

Schmieder, R., Edwards, R. (2011, March 9). Fast identification and removal of sequence contamination from genomic and metagenomic datasets. *PLoS ONE,* 6(3):e17288. doi: 10.1371/journal.pone.0017288.

Sequencing: Implications for Sequence-Based Analysis of Clinical Samples. *PLoS Pathog*, 10(11): e1004437. doi: 10.1371/journal.ppat.1004437.

Siragua, E., Weese, D., Reinert, K. (2013, April) Fast and accurate read mapping with approximate seeds and multiple backtracking. *Nucleic Acids Research. 41(7): e78. doi: 10.1093/nar/gkt005*

Strong, M. J., Xu, G., Morici, L., Bon-Durant, S. S., Baddoo, M., Lin, Z., Fewell, C., Taylor, C. M., Flemington, E. K. (2014, November). Microbial Contamination in Next Generation