Methods of Contaminant Detection in Simulated Sequencing Reads

Eleanor Hilgart, Matthew Ost, Gabe Shatkin, & Alexey Solganik

Abstract

We developed a tool that searches for the presence of contaminants such as mycoplasma, bacteria, or fungi in sequencing data. Using known contaminant genomes from public databases and searching the sequencing data for those contaminant genomes, indicating the presence of the contaminant in the sample, we investigated both online and offline methods of preprocessing the contaminant query sequence or the sequencing data being queried. Online methods included the Smith-Waterman algorithm to match sequencing reads to contaminant reference genomes. For offline approaches, we built different data structures, such as a k-mer and FM index, over the contaminant genomes. We also experimented with MinHash as a way to efficiently compare sequencing reads to the contaminant genomes. All methods proved useful for analyzing small datasets, but encountered significant issues in time and space requirements when scaling to larger datasets.

Introduction

Contamination in sequencing datasets is an issue of interest. Samples can be contaminated by microbes in the laboratory, laboratory personnel, contamination in kits, and other modes, leading to the presence of contaminant DNA in the sequencing data (De Simone et al., 2020; Zhang et al., 2020). This can affect the results of many studies by causing incorrect conclusions about gene function and taxonomy, creating bias in genomic analysis, along with other effects (Francois et al., 2020). Incorrect conclusions from contaminated data can also easily impact new analyses, and thus be passed between databases, amplifying the impact of the contamination (Francois et al., 2020). These issues can be particularly detrimental in microbiome

analysis, as the presence of exogenous microbes can significantly affect conclusions of community analyses (Salter et al., 2014). It is unknown exactly how prevalent contamination is in large genomic datasets; one study found that up to 8% of coding sequences in an arthropod sequence database could have originated from contaminants, with 35% of the published genomes showing evidence of contamination (Francois et al., 2020). Identifying and removing contamination in sequencing datasets is important to improve the validity of sequencing results and analysis.

Here, we present a tool to identify contamination in sequencing reads. Reads are aligned to a human reference genome or a contaminant genome from an internal database. Reads aligning to contaminant genomes are identified and removed from the data, returning a decontaminated data file. We offer several methods of alignment: k-mer index, FM index, or Smith-Waterman; additionally, we offer MinHash as an alternative method of identifying contamination in reads. These were chosen as representative alignment strategies that would allow us to identify contamination; MinHash was chosen because it is a highly efficient method for comparing large texts. We compare the performance of these different methods, as well as comparing to previously published tools for decontamination.

Previous Work

Many tools have previously been created to address the issue of contamination in sequencing datasets using a wide variety of approaches. These can be broadly classified as two different strategies: by alignment or by a probabilistic model. DecontaMiner, DeconSeq, and FastQ screen fall in the former category. DecontaMiner (Sangiovanni et al., 2019) identifies microbial sources of contamination by aligning any unmapped reads to contaminant genome databases using MegaBLAST, which implements a greedy dynamic programming method of

alignment (Zhang et al., 2000). DeconSeq uses a combined Burrows-Wheeler aligner (BWA) with Smith-Waterman to identify and remove human contamination from bacterial and viral metagenomes, which is very important in microbiome analyses (Schmieder & Edwards, 2011a). FastQ screen implements either Bowtie or BWA to map reads to different genomes (Wingett & Andrews, 2018). In contrast, Recentrifuge uses a statistical approach to probabilistically classify taxa found in a multi-species sample as contaminant or native, using read-by-read taxonomic classification from other software to construct confidence levels for the presence of each taxa in the sample (Martí, 2019). PRINSEQ, a tool for filtering and cleaning sequence data, allows for the identification of contamination based on odds ratios of the presence of different dinucleotides from reads or an assembled genome (Schmieder & Edwards, 2011b). Our approach was inspired by the read alignment strategies seen in DecontaMiner and FastQ screen.

Methods

Data Generation

The CHM1_1.1 and GRCh38.p13 human genomes were obtained from NCBI RefSeq for use as the human reference (O'Leary et al., 2016). Initially, 31 bacterial and fungal contaminant genomes were obtained from the NCBI Genome datasets (NLM & NCBI, 2004); these were chosen as prevalent contaminants that previous literature had found in sequencing datasets (Sangiovanni et al., 2019; Supplementary Table 1). The database was later reduced from these initial 31 contaminants and two human genomes to the CHM1_1.1 V1 genome and 8 contaminants for better tractability (Supplementary Table 2). Data were cleaned by removing long sequences of no-confidence bases (Ns) to allow for read generation. Contaminated and uncontaminated reads were simulated using InSilicoSeq (Gourlé et al., 2019). This tool takes an input of FASTA files and outputs paired-end simulated reads in FASTQ format; it also simulates

quality scores and sequencing errors based on kernel density estimator models produced from Illumina sequencer data. One can additionally specify the desired abundance of the various input genomes in the output FASTQ. Various sizes of FASTQ files with 150 bp reads were generated with 70-80% human contents and different distributions of contamination, ranging from a single bacterial contaminant to eight different contaminants.

One major difficulty encountered when using InSilicoSeq was our initial inability to produce reads with no errors. Considerable time was spent attempting to create our own no-error model; this process required inputting a BAM file with perfectly aligned reads to the model generation function of InSilicoSeq. We attempted to make our own synthetic perfectly aligned BAM files using Bowtie2 and Samtools. However, the error models created this way resulted in errors when used as an input to the InSilicoSeq data generation function. Eventually, we discovered a pre-built no-error function in the InSilicoSeq code that had been excluded from any available documentation or user guides. Once this was realized we were able to simulate FASTQ files with and without sequencing errors.

Data Utils and Evaluation

To support our methods, we also developed some utility classes and functions for handling FASTA and FASTQ files efficiently. We created custom Python objects that support efficient iteration and indexing of these data files, providing an intuitive and--most importantly--consistent abstraction for working with these files in our code. Since many of our bacterial and fungal reference genomes are multi-FASTA files, our FASTA objects support this format by default. Internally, the data is represented as a list of each fragment in the file. Note that building FASTA objects over the entire human genome can take up to 60 seconds, since the files are extremely long. Lastly, we developed some simple code to save our indexes to disk so that they

can be computed offline and re-used at runtime. Time was reported as wall clock time with the Unix time command. Memory was evaluated using the Fil memory profiler (Hyphenated Enterprises LLC, 2021).

K-Mer Index

The most basic method for contaminant detection we applied was a k-mer index. This method first analyses the provided references offline, creating a dictionary that stores the start location of all k-mers of a specified length in each reference genome-- provided as FASTA file inputs. The reads to be analyzed are input as FASTQ files. Our code iterates through each read in the FASTQ file, creating a list of all k-mers in the read. Reads are then filtered based on how many k-mers in the read are found in each reference index. Cutoffs for the amount of required matches are based on a tolerance for mismatches input by the user and a modified pigeonhole principle. Reads that passed this filter are compared base-by-base to the potential start sites in the reference suggested by the indices where the first k-mer in the read is found in the reference. Any reads that align with a reference with fewer mismatches than the input tolerance are assigned to that reference. Reads are first aligned to the desired reference and then to the contaminant references in the order they are input. Reads are only allowed to be assigned to one category, either "desired", "contaminant", or "unassigned". Our code outputs three lists, containing the read numbers that were assigned to each category. This allows the user to either use only reads aligned to the desired reference or to discard a sequencing run if contamination is found.

FM Index

FM-Index is an offline string preprocessing method that utilizes the Burrows-Wheeler Transform (BWT) for storage and partial Suffix Array (SA) for reference querying. First, the

BWT is constructed using simple methods for rotations and sorting. Next, a tally matrix is built to store the indices of the characters in BWT to enable querying using LF-mapping. There are two problems that arise when using this matrix: if the reference sequence is long, then storing *O(m)* integers (*m* being the length of the sequence) takes a lot of space, and scanning through *O(m)* is time costly. We solved both of these problems by only including certain "checkpoints" with specified distance in the tally matrix. By doing so, we were able to significantly diminish the number of integers we had to store. The lookup time became constant because only two indices calculations are needed in addition to a small linear search between these indices, the length of which we can control by modifying the distance between the "checkpoints". Lastly, we implemented a SA and also removed a lot of its entries to save space. If a certain index in the BWT did not have a corresponding SA entry, we used LF-mapping to find a BWT index from which we were able to calculate the data of the original SA entry. Unfortunately, for this method we were only able to implement exact matching.

Smith-Waterman

Smith-Waterman is a dynamic programming algorithm that creates a matrix H of alignment scores between substrings of two strings X and Y (Smith & Waterman, 1981). Alignment scores are calculated based on a penalty function giving a negative score to mismatches and gaps and a positive score to matches. Each cell $H_{i,j}$ contains the maximum score of any pair of substrings ending at X_{i-1} and Y_{j-1}, including the empty strings which are defined to have alignment score of 0. The matrix H has size $|X| \times |Y|$. Our implementation takes a read X and a reference genome Y, creates the dynamic programming matrix, and finds the maximum score in the last row of the matrix corresponding to the maximum alignment score of the whole read to any substring of the reference genome. The read is aligned to every reference genome,

and the best alignment for that read to every reference is taken. If the best alignment is to a contaminant genome, the read is classified as contaminant; if the best alignment is to a human chromosome, the read is classified as human; else, the read cannot be assigned. The function returns a dictionary containing lists of the read IDs with their designations. We initially considered integrating Smith-Waterman with the FM index as described in DeconSeq (Schmieder & Edwards, 2011a), but further investigation revealed that this BWA-SW aligner is optimized for long reads, rather than the 100-150 bp reads of our own data. Thus, Smith-Waterman was implemented naively.

MinHash

MinHash, as proposed by Broder (1997), is a way of creating sketches of text documents through hashing. These sketches are extremely small and enable very efficient similarity computations between large documents. In the context of genomics, we create a document of words out of a genome by breaking the DNA string into k-mers. As described in the Mash paper (Ondov et al., 2016), the sketch of the genome is created by hashing each k-mer and keeping the smallest M hash values. MinHash sketches are often compared to one another using an error-bounded estimation of Jaccard similarity, which Broder describes as *resemblance*. Broder also describes *containment*, an estimation of how closely one document is contained in another. We primarily use containment in our contaminant detection pipeline. Both of these metrics are scalar values between 0 and 1.

We first create sketches of each reference genome, including the human genome and all bacterial and fungal contaminants in our database. We use a k-mer size of 21 and a sketch size of 1000, as suggested by Ondov et al. (2016). Constructing MinHash sketches can be done in linear time with the length of the text, however the sheer size of the human genome acts as a significant

bottleneck during online processing. To overcome this, we create and save to disk the sketches for each reference genome, thus ensuring they do not need to be re-computed for each contaminant detection process. Updating each reference sketch in a stream instead of generating a list or set of all k-mers ensures that very little RAM is needed, even when working with the human genome and its 3 billion nucleotides. For reference, the total disk space needed to store MinHash sketches for the entire human genome in our database is under 2 MB, making this a highly memory-efficient method. To identify contaminated reads, we create a MinHash sketch of each read and compute its containment score with each reference genome. If the reference genome with the highest score is a contaminant, the read will be labeled accordingly.

Results

K-mer Index

Although simple, the k-mer index was very successful for small datasets-- when reference genomes were only thousands of bases in length and FASTQ files had <10,000 reads. For these small datasets, the k-mer index method was 100% percent accurate when identifying reads from the desired reference and over 90% accurate when identifying contaminated reads from multiple contaminant sources. The remaining reads that the method failed to identify as contaminant were always unassigned and never attributed to the desired reference. The k-mer index had severe limitations when applied to larger datasets. When reference genomes were the size of one human chromosome or larger, memory and size became an issue. K-mer indices are comparable in size to the reference genomes and run times to construct a k-mer index on a personal computer became unreasonable for the scope of this project-- code was still running after multiple hours. Even with reference genomes on the order of thousands of base pairs, when more realistic numbers of reads were used-- 500,000 or greater-- our code took at least a half

hour to run. This time limitation will only increase proportionally to the size of the index that has to be searched through in order to find alignments. This shows that the undesirable time requirements of the k-mer index method are due to both the offline and online components, meaning that the issues will not be resolved by a one-time time cost to create indices of reference genomes. Both on and offline time and memory issues could be addressed by computers with higher computing power, but this is an impractical approach, as we will see by the better performance of the other methods explored in this project.

FM Index

The FM Index method had by far the lowest observed contaminant percentage for the small data sets, which was expected since it was implemented as an exact matching algorithm (Table 1). The observed contaminant fraction was approximately 3 times lower than the other methods, which serves as a good demonstration of the significant advantage of approximate matching methods. As for the large data sets, no data was collected since the algorithm ran for too long. FM-index time and space complexities are also unimpressive when compared to other offline methods like K-mer Index and MinHash. The explanation for that is that this method consists of many pieces that proved to be hard to implement. A lot of these pieces can be optimized to work faster and occupy less memory. For example, we are interested in exploring more efficient implementations of the Burrows-Wheeler Transform. This data structure can be built faster than in our implementations and also can utilize compression techniques to save a lot of space. Another major drawback is that we had to generate whole Suffix Arrays before discarding most of their elements, which means that during the construction phase of the FM-index we were storing $O(m)$ integers. Thus, although it has shown to be the weakest offline algorithm out of the ones we tested, we are interested in continuing investigating FM-index and

the next steps in improving it will be optimizing Burrows-Wheeler Transform time and space complexities, reconsidering partial Suffix Array building algorithm and implementing the approximate matching version of FM-index.

Smith-Waterman

The Smith-Waterman algorithm was 100% accurate when run on very small data with around 50 reads with only one contaminant and one chromosome as references, with no unassigned reads, although it took over one minute for this small data (Table 1). However, when run on the same data with another contaminant reference added, it timed out. We did not attempt to run the algorithm on the largest datasets created from the entire human genome and many contaminants because of this. The time issue is likely because of the many large matrices that are created in this implementation -- each matrix is $n \times m$ where n is the size of the read and m is the size of the reference genome. Clearly, these matrices can grow very fast as the size of the reference genome increases; additionally, every read has one such matrix per human chromosome and one per contaminant reference used. These extremely large dynamic programming matrices over the reference genomes make the naive strategy untenable for read alignment against large references. Nevertheless, our implementation allows us to accurately find contamination over smaller read sets created from limited references. A more efficient way to implement this would be to introduce some filtering strategy to reduce the search range of the dynamic programming matrices in a seed and extend method.

MinHash

As expected, the MinHash algorithm proved to be space-efficient and computationally tractable, even with large reference genomes and many sequencing reads. The largest example we ran consisted of 6 bacterial genomes, 3 fungal genomes, and the entire human genome as

references with our largest file of 50,000 synthesized sequencing reads. This took roughly 35 minutes to run, which we believe can be further decreased through some simple multi-core parallelism. However, this method was not particularly effective at identifying contaminant reads on real data. It performed perfectly on a small data example, but its predictions degenerated when multiple large reference genomes were used. In the aforementioned large example, the model classified 0.01% of reads as contaminated, which is much lower than the expected threshold of 24%. Upon closer inspection, it was common for a read sketch to have extreme containment scores (close to 0 or close to 1) with both contaminant and human reference sketches, depending whether cont(read, ref) or cont(ref, read) is being computed. These scores seem too unrealistic, which indicates that the sketches as currently constructed are not sufficient for contaminant identification. We speculate that this instability is due to the vast size difference between individual reads (100-150 nucleotides) and reference genomes, which results in a higher estimation error. In short, these sketches are not sufficient for encoding the underlying essence of each read. More work is needed to uncover the true cause of this issue, however. The issue persists when testing on error-free synthetic reads, so it is not simply an issue of sequencing errors. Regardless, it may eventually prove useful to add an additional processing layer to correct errors or remove low-frequency k-mers.

Table 1. Performance comparison between K-mer index, FM index, Smith-Waterman, MinHash, and DecontaMiner (Sangiovanni et al., 2019). Our programs were evaluated on the smallest data constructed from a cut single chromosome (chromosome 1) and a single cut contaminant (*M. fermentans*). Files were as follows: reads: chr1_Mfermentans_8020_hiseq_reads_tinycut_R1.fastq; references: noN_chr1_cut.fasta, Mfermentansbac1_cut.fasta. Note that as this is a tiny test case, the speed and memory may grow very fast for some methods as the size of the references and number of reads grow.

	Expected Contaminant %	**Observed Contaminant %**	**Speed (s)**	**Online Memory Consumption (MiB)**
K-mer index *	20%	18%	0.234s	1.3 MiB
FM Index	20%	6%	0.596s	22.7 MiB
Smith-Waterman	20%	20%	73.835s	11.4 MiB
MinHash	20%	20%	0.593s	0.6 MiB
DecontaMiner**	22.5%	15.7%	Not Available	Not Available

* with a tolerance allowing 5 mismatches; ** DecontaMiner performance data obtained for different dataset from Sangiovanni et al. (2019)

Conclusions

In this project, we explored several different methods of contaminant detection over a limited database of contaminant genomes. The methods generally performed well on small datasets, but had difficulty with scaling to realistic data sizes. The k-mer index was the fastest performing method on small datasets, but became unreasonably slow for more practically sized datasets; memory will also theoretically grow proportionally to the reference genome size which is undesirable for this method. Smith-Waterman had high accuracy, but was unable to be run on large datasets that are close to real-world sizes. FM-index was the algorithm with the worst space-complexity and accuracy, but has a lot of room for improvement with more advanced implementation techniques. While highly time and space efficient, MinHash was inaccurate

when run on large data with multiple reference genomes. Future directions of investigation include extending the reference database, integrating a filtering method with Smith-Waterman, introducing compression and approximate matching into FM-index, parallelizing the contaminant identification process, and exploring ways to make MinHash sketches of reads more robust.

Literature Cited

Broder, Andrei. (1997). On the Resemblance and Containment of Documents. Proceedings of the International Conference on Compression and Complexity of Sequences. 10.1109/SEQUEN.1997.666900.

De Simone, G., Pasquadibisceglie, A., Proietto, R., Polticelli, F., Aime, S., J M Op den Camp, H., & Ascenzi, P. (2020). Contaminations in (meta)genome data: An open issue for the scientific community. IUBMB life, 72(4), 698–705. https://doi.org/10.1002/iub.2216

Francois, C. M., Durand, F., Figuet, E., & Galtier, N. (2020). Prevalence and Implications of Contamination in Public Genomic Resources: A Case Study of 43 Reference Arthropod Assemblies. G3 (Bethesda, Md.), 10(2), 721–730. https://doi.org/10.1534/g3.119.400758

Gourlé, H., Karlsson-Lindsjö, O., Hayer, J., & Bongcam-Rudloff, E. (2019). Simulating Illumina metagenomic data with InSilicoSeq. Bioinformatics (Oxford, England), 35(3), 521–522. https://doi.org/10.1093/bioinformatics/bty630

Hyphenated Enterprises LLC. (2021). Fil Memory Profile [Computer software]. Retrieved from https://pythonspeed.com/fil/docs/index.html

Martí J. M. (2019). Recentrifuge: Robust comparative analysis and contamination removal for metagenomics. PLoS computational biology, 15(4), e1006967. https://doi.org/10.1371/journal.pcbi.1006967

National Library of Medicine (NLM; US), National Center for Biotechnology Information (NCBI; 2004). Genomes – NCBI Datasets. NCBI. https://www.ncbi.nlm.nih.gov/gene/

O'Leary, N. A., Wright, M. W., Brister, J. R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., Astashyn, A., Badretdin, A., Bao, Y., Blinkova, O., Brover, V., Chetvernin, V., Choi, J., Cox, E., Ermolaeva, O., Farrell, C. M.,

… Pruitt, K. D. (2016). Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. Nucleic acids research, 44(D1), D733–D745. https://doi.org/10.1093/nar/gkv1189

Ondov, B.D., Treangen, T.J., Melsted, P. et al. Mash: fast genome and metagenome distance estimation using MinHash. Genome Biol 17, 132 (2016). https://doi.org/10.1186/s13059-016-0997-x

Salter, S. J., Cox, M. J., Turek, E. M., Calus, S. T., Cookson, W. O., Moffatt, M. F., Turner, P., Parkhill, J., Loman, N. J., & Walker, A. W. (2014). Reagent and laboratory contamination can critically impact sequence-based microbiome analyses. BMC biology, 12, 87. https://doi.org/10.1186/s12915-014-0087-z

Sangiovanni, M., Granata, I., Thind, A. et al. From trash to treasure: detecting unexpected contamination in unmapped NGS data. BMC Bioinformatics 20, 168 (2019). https://doi.org/10.1186/s12859-019-2684-x

Schmieder, R., & Edwards, R. (2011a). Fast identification and removal of sequence contamination from genomic and metagenomic datasets. PloS one, 6(3), e17288. https://doi.org/10.1371/journal.pone.0017288

Schmieder, R., & Edwards, R. (2011b). Quality control and preprocessing of metagenomic datasets. Bioinformatics (Oxford, England), 27(6), 863–864. https://doi.org/10.1093/bioinformatics/btr026

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. Journal of molecular biology, 147(1), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

Zhang, F., Flickinger, M., Taliun, S., InPSYght Psychiatric Genetics Consortium, Abecasis, G. R., Scott, L. J., McCaroll, S. A., Pato, C. N., Boehnke, M., & Kang, H. M. (2020). Ancestry-agnostic estimation of DNA sample contamination from sequence reads. Genome research, 30(2), 185–194. https://doi.org/10.1101/gr.246934.118

Zhang, Z., Schwartz, S., Wagner, L., & Miller, W. (2000). A greedy algorithm for aligning DNA sequences. Journal of computational biology : a journal of computational molecular cell biology, 7(1-2), 203–214. https://doi.org/10.1089/10665270050081478