# ReadSniper: Rapid Retrieval of Relevant Reads for Discovering Relatives of Viruses

Anton Oresten Sollman

August 4, 2022

**Abstract**

Viruses can exhibit staggering diversity, and it is not uncommon to discover a virus in a sample that bears little resemblance to anything found with standard tools and databases. The largest collection of next generation sequencing data, the Sequence Read Archive (SRA) represents a potential source of related viruses, but computational limitations do not allow us to directly query the entire SRA with a given virus sequence. For RNA viruses, a recent effort (Serratus.io) has indexed the entire SRA using only the RdRp signatures of RNA viruses. Given an RdRp sequence, we can retrieve the datasets that contain close relatives.

Here we propose to develop a tool that leverages this Serratus.io indexing, and attempts to further mine these SRA datasets for regions outside of just the RdRp, identifying and retrieving all reads that are sufficiently similar to a query virus genome. We use a strategy that aims to efficiently identify reads for which the component k-mers of the read are enriched within a short window of the query genome. Once these reads have been retrieved, they can be assembled into related genomes against which the evolution of the query virus can be assessed.

## 1 Introduction

New methods and techniques are crucial for tackling the increasing amount of biological sequence data. Tools such as BLAST do not always return a match, which pushes a need for creative and innovative ways to find relatives of a genome in public datasets.

A project by the name of Serratus[9] attempts to uncover the planetary virome by re-analyzing petabytes of public data[2]. Using a query virus' RNA-dependent RNA polymerase (RdRp)-sequence, Serratus can find datasets from the Sequence Read Archive (SRA)[8] that may contain relatives of the query virus genome. This speeds up the process of finding relevant datasets dramatically.

The SRA is NCBI's (National Center for Biotechnology Information) public bioinformatics database that provides a public repository for DNA sequencing data, especially the short reads generated from high-throughput sequencing, which are typically less than 1,000 base pairs in length. SRA datasets come in FASTQ format. A FASTQ file stores a list of nucleotide sequence *reads* (synonymous to *records* or *spots*). The FASTQ format also contains information about the quality of each base, and the length of each read.

The method that we are presenting heavily relies on the use of k-mers, which are contiguous nucleotide subsequences of length $k$. The term *k-mer* shares its root with *polymer*, and simply means *k parts*. The benefit of using k-mers is that we can directly look for exact matches, instead of approximate matches.

## 2 Method

Serratus, when given an RdRp sequence, will go through its database and return a list of datasets containing similar RdRp sequences, along with some values for each dataset. One of these values is called *pident*, which is the percentage of RdRp base pairs that are identical in relation to the query RdRp sequence. Another value is *coverage*, which tells us how much of the RdRp sequence was found in a dataset. The datasets with the highest pident and coverage get selected for processing.

Datasets are pulled from the SRA using SRA Toolkit[11]. The `fastq-dump` command is used with the `--split-files` argument, which will write paired reads (necessary for later assembly). Not using the `--split-files` argument will result in the paired reads being concatenated into single reads.

The Julia Programming Language was used for this project because of its speed and native multiple dispatch[1]. The Julia community has developed several packages which aid in the process of analyzing large amounts of biological sequence data.

The query genome, or reference, is stored in a FASTA file, and is read using the FASTX Julia package[5]. FASTX creates a LongDNASeq-type sequence, which uses four bits to represent each base pair (allowing for ambiguous bases). The LongDNASeq type comes from the BioSequences Julia package[4], and is a more suitable type to use than Julia's built-in String. The LongDNASeq was also used to store k-mers, even though BioSequences has its own immutable Mer type.

## 2.1 Matching k-mers

An appropriate k-value needs to be used. If k is too small, there will be a lot of spurious matches arising because the number of k-mers quickly exceed the number of possible k-mers with a given k, which is $4^k$. If, however, k is too large, then random mutations and deviations from the query will block a lot of exact matches.

**Figure 1:** This alignment illustrates a small section of the query and a read which have a clear correlation. Due to a minor mutation, however, the matching section could go unnoticed if k is too large. This is a consequence of only searching for exact matches.

A constant look-up time for finding k-mer matches with the query genome was achieved using a hash table, or dictionary. The hash table takes the k-mers as keys, and an integer vector of indices as values, representing where the k-mer occurred in the query.

| k-mer | Indices |
|---|---|
| ACGT | 1, 5 |
| CGTA | 2 |
| GTAC | 3 |
| TACG | 4 |

**Table 1:** The k-mer hash table for "ACGTACGT" (with a $k$ of 4)

While iterating through the reads of a dataset, the hash table is used to see if and where k-mers from reads had a match with the query. If no match is found for a k-mer, an empty integer vector is returned. For every read, the index vectors that are returned get concatenated into one vector containing all of the k-mer matches.

There are $L-k+1$ contiguous subsequences of length $k$, for a read of length $L$. Not all k-mers need to be checked, however. A step size can be set, meaning that we only use every nth k-mer. A step size of 3 (instead of 1), for example, would result in a third the amount of k-mers, meaning that there would be approximately a third the amount of matches. The step size is therefore inversely proportional to the number of k-mers that have to be compared to the query. Matching k-mers with the query takes up the majority of the runtime, meaning that the step size would also be inversely proportional to the time it takes to run the script. Skipping k-mers does work, and is valid as long as we later account for the reduction in matches and sample size.

## 2.2 Reads with clusters of matches

These vectors will contain some spurious matches. If there are true matches however, we would expect most of them to be in close proximity of each other in the query genome. The range of one cluster will practically only be as long as the read itself.

The vector is sorted so that clusters can be spotted more easily, since they would then appear as straight lines when plotting against the position in the query. The matches need to be regular, and not disturbed by mutations for the line to appear straight.

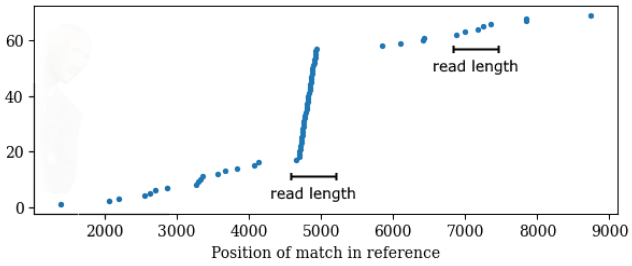


**Figure 2:** A scatter plot, showing k-mer matches with the reference sequence. A cluster of k-mer matches indicate that a region of a read is matching to a region of the reference sequence.

A way of finding the maximum number of points within a read-length-sized sliding range was needed in order to find the densest cluster in each read. See Figure 2, and note that the size of the range, or window, does not change. The size of the window is not dependent on the number of matches, but instead has to do with the distance from the first and last position in the reference.

One solution to this problem has a time complexity of $\mathcal{O}(n \log n)$. It works by iterating through all the matches, testing all possible start indices, and then performing a binary search to find the last possible end index that fits in the range. The condition used for the binary search differs from the usual, however. Instead of the condition being whether the middle value is higher or lower, the condition is whether the match position is inside or outside of the range. Be aware that a faster, $\mathcal{O}(n)$ solution exists.

## 2.3 Threshold for cluster size

Reads are assigned individual scores based on size of the largest cluster compared to the read length. Some reads are naturally going to be shorter than others, and the number of k-mers (as well as matches) in a read will therefore be proportional to the read length. Hence, we divide by the length to get an unbiased score. This roughly translates to the density of matches, inside of the window with the maximum number of matches.

Multiplying by the step is also necessary, as it is inversely proportional to the number of matches, and therefore the density of the largest cluster.

$$\text{score} = \frac{\text{max cluster}}{\text{read length}} \cdot \text{step}$$

The score will be between 0 and 1, assuming that the number of spurious matches is not too high because of a low $k$-value.

The scores were plotted onto a histogram (See Figure 3), which gave us an idea of what an appropriate threshold would be, in order to filter out most of the background noise.

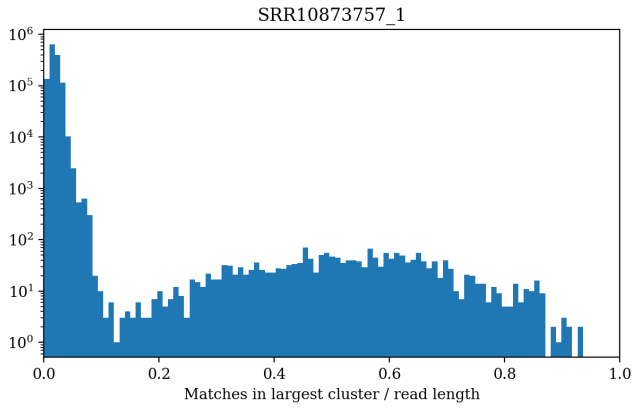


**Figure 3:** A histogram showing the distribution of cluster densities for reads in a dataset. The spike on the far left comes from reads where the largest cluster is only a small portion of the read length, and therefore likely only contains spurious matches. Note that the y-axis is logarithmic.

### 2.4 Skyline of Matches with the Query

An overview of matches to the query is important for assessing how much of the genome could be assembled using a dataset. Ranges were created using the first and last index of matches with the query above a certain threshold. Those ranges were then added into an array, and plotted onto a histogram (See Figure 4), with the number of bars equaling the length of the query.

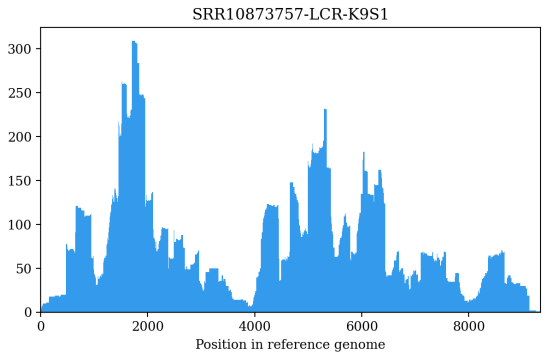


**Figure 4:** A histogram showing how many matches there are to which parts of the query sequence.

The histogram is referred to as a skyline graph because of it looking like a city skyline.

### 2.5 Genome Assembly

The last step is genome assembly, which is done using a public genome assembly toolkit called *SPAdes*[10]. It is used to assemble a genome similar to the query, using reads with a score that exceeded the threshold, for each dataset.

There were alternatives to SPAdes, such as the GenomeGraphs Julia Package[6]. We attempted to use GenomeGraphs before settling for SPAdes. Both required the datasets to be split using the `--split-files` argument in the `fastq-dump` command. This also required a slight change in the code. The algorithm had to run two datasets in parallel (although not necessarily at the same time), and then merge them into a filtered dataset. The IDs of reads that passed the threshold requirement were put in a CSV file. A script was made which would iterate through the paired datasets and write reads whose IDs were in the CSV file into a new filtered FASTQ file.

## 3 Results

We were able to construct versions of our query genome using reads with scores that passed a threshold of around 0.1. We tried to construct one genome per dataset; there were a few that had a wide enough range of matches to make that possible. By comparing the similarity of the genomes that were assembled, a phylogenetic tree could be constructed (See Figure 5).

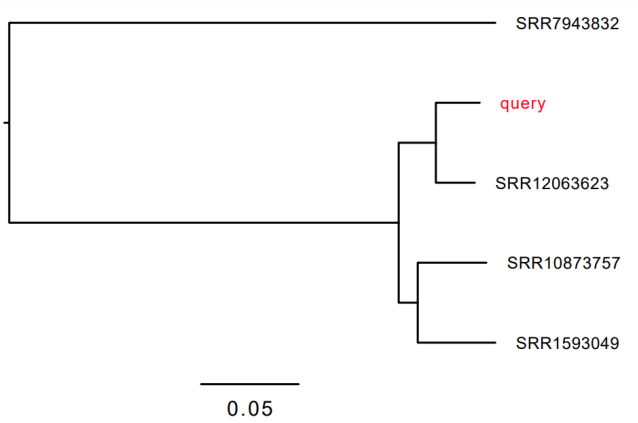


**Figure 5:** A phylogenetic tree that shows how close the assembled genomes are to each other, as well as the query.

The time it took to process one dataset was roughly proportional to its size, but usually between 10 minutes to an hour. The time can be decreased by increasing the step size. SRR7943832 (a dataset from the SRA) is further away because it has the lowest pident, at 82.4, whereas the other datasets that were used had a pident of 98.3.

## 4 Discussion

These datasets come from a wide variety of samples. SRR1593049 for example, came from "Ballast water samples from five bulk carriers from different parts of the Great Lakes"[13], while SRR10873757 came from "Viral metagenomics of anal swab samples from wild and zoo birds in China"[12]. The fact that we were able to assemble our query, which came from a fecal sample from a patient who had an infection, with datasets from such diverse samples is incredibly fascinating. Assembly of the genome using more datasets is necessary to truly assess the phylogenetic tree and what our virus could be.

### 4.1 Improvements

There is still plenty of room for improvement. An algorithm for assigning scores to reads with a time complexity of $\mathcal{O}(n)$ does exists, but has not yet been implemented. This didn't matter too much in this scenario, as it's a small portion of the total time, and sorting the vector is already $\mathcal{O}(n \log n)$.

The metric used for scoring could be changed to something other than the *number* of matches in a window. We could for example keep track of the read positions of the matches, and rearrange it the same way we rearrange the elements when we sort the query positions of the matches. We could then calculate the size of the maximum increasing subsequence within each window, and use that size as a metric instead of the number of matches. Doing this would decrease the average score for reads with no correlation to the query, and thus, the background noise gets moved a little more to the left (See spike in 3), which lets us lower the threshold.

Another technique that could be utilized for getting more matches and expanding the tree is to use the new assembled genomes to create a longer reference genome. The genomes could for example be concatenated, and made into a super-query. This, coupled with also searching using the genome's reverse complement, should ensure that we miss as few reads as possible.

Two bits can be used for storing biological sequences. The reason why we use four bits is because some datasets can have ambiguous nucleotides, such as N, which would return an error if only two bits were to be used.

Since score is inversely proportional to read length, the threshold should be adapted based on the read length. The background noise of datasets with longer reads will have a lower average score, meaning that the spike to the left in Figure 3 will have varying fatness, depending on the average read length.

### 4.2 Future work

We hope to make an easy-to-use package for Julia, that takes a query genome, and returns a list of datasets, coupled with assembled genomes from those datasets. The source code for this project will be publicly available on GitHub[7].

Other tools exist for retrieving sets of reads related to a query sequence. Many of these are for "mapping", which addresses both the read identification problem and the alignment of the reads to the query genome. Since our downstream applications will involve contig assembly, we do not need to align the reads to the query genome. It is therefore possible that we can solve just the read retrieval problem faster than approaches that perform sequence alignment, such as NextGenMap[3], and we will investigate this.

## 5 Acknowledgements

I would like to sincerely thank my supervisor, Ben Murrell, for guiding me through all the concepts that made this project possible.

I am grateful to Karolinska Institutet for giving me the opportunity to be a part of the Summer Research School in Bioinformatics and Computational Biology. This would also not have been possible without its coordinator, Gerry McInerney.

## 6 References

[1] Bezanson et al. *Julia: A Fast Dynamic Language for Technical Computing*. URL: https://arxiv.org/abs/1209.5145. (accessed: 2022-08-03).

[2] Edgar et al. *Petabase-scale sequence alignment catalyses viral discovery*. URL: https://www.nature.com/articles/s41586-021-04332-2. (accessed: 2022-07-29).

[3] Sedlazeck et al. *NextGenMap: fast and accurate read mapping in highly polymorphic genomes*. URL: https://academic.oup.com/bioinformatics/article/29/21/2790/195626. (accessed: 2022-07-29).

[4] *BioSequences*. URL: https://biojulia.net/BioSequences.jl/stable/.

[5] *FASTX*. URL: https://biojulia.net/FASTX.jl/stable/.

[6] *GenomeGraphs*. URL: https://biojulia.net/GenomeGraphs.jl/stable/.

[7] *ReadSniper: Relevant Read Retriever*. URL: https://github.com/Periareion/ReadSniper.

[8] *Sequence Read Archive*. URL: https://www.ncbi.nlm.nih.gov/sra.

[9] *Serratus*. URL: https://serratus.io/.

[10] *SPAdes*. URL: https://github.com/ablab/spades.

[11] *SRA Tools*. URL: https://github.com/ncbi/sra-tools.

[12] *SRR10873757*. URL: https://www.ncbi.nlm.nih.gov/sra/?term=SRR10873757.

[13] *SRR1593049*. URL: https://www.ncbi.nlm.nih.gov/sra/?term=SRR1593049.