Towards Petabyte-scale k-mer Counting with Universal Frequency Minimizers

Johan Nyström-Persson (johan@jnpsolutions.io), JNP Solutions, Sumida-ku, Tokyo, Japan
Gabriel Keeble-Gagnère, Agriculture Victoria, AgriBio, Centre for AgriBioScience, Bundoora, VIC, Australia

K-mer counting is a basic technique for analysis of genomic sequences. We previously introduced Discount[3], a highly scalable distributed k-mer counter based on Apache Spark, a modern big data framework. Here we describe how we managed to k-mer count a 6 TB metagenomic RNA dataset, with a total of 6 trillion k-mers, and speculate about how the technique might be scaled up to petabytes, enabling the counting of quadrillions of k-mers.

K-mers

K-mers are genomic subsequences of length *k* (typically 20-50, but in this work unbounded), which have many applications, including genome assembly, read binning, and metagenomic profiling. The most basic k-mer analysis is counting each distinct k-mer and producing a table of k-mer counts. Here, we study k-mer counting at extremely large scale as a gateway to enabling other kinds of genomic analysis at that scale. Below is a toy example for k = 10, where a read has been split into overlapping 10-mers.

```
TGGCGTCATTTTCCCCAATCATAGACTGTTGAAAG…
TGGCGTCATT
 GGCGTCATTT
  GCGTCATTTT
   CGTCATTTTC
    …
                ATAGACTGTT
                  …
```

Minimizers

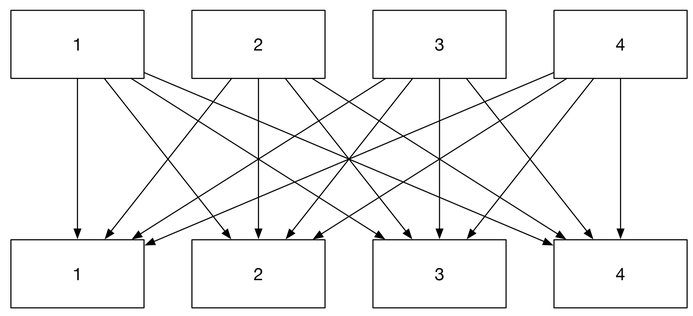
Minimizers are substrings of length m < k for some m, that can be used to split a read into *super-mers* (super k-mers, or k+x-mers). They are the minimal m-mer in each k-mer for some ordering of m-mers ("minimizer ordering"). Super-mers are consecutive k-mers that share the same minimizer.
Long super-mers also help data compression, as a compact representation of their k-mers. Below is an example for k=5, m=10, with minimizers highlighted in blue.
Minimizers can also be used to bucket k-mers and super-mers, since a given k-mer will always have the same minimizer.

```
        TGGCGTCATTTTCCCCAATCATAGACTGTTGAAAGTGAACA…
        TGGCGTCATT
         GGCGTCATTT
          GCGTCATTTTCCCCA
                TTTTCCCCAA
                 TTTCCCCAAT
                  TTCCCCAATC
                   TCCCCAATCATAGAC
                         ATCATAGACTG
                            CATAGACTGTTGAAA
                                  CTGTTGAAAG
                                   TGTTGAAAGTGAACA
```

Apache Spark - a big data framework

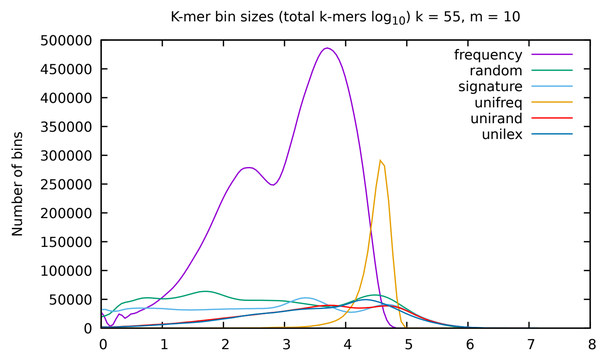
As the size of genomic datasets continues to grow, some analyses will only be possible by using a cluster rather than a single machine. In recent years, modern big data frameworks have solved many problems related to parallel processing and enjoy broad support from commercial cloud providers, as well as being easy to run on a single machine. This allows us to think less about technical problems like file formats and thread management, instead focusing on the core bioinformatics problem. In this work we use the Apache Spark framework, and particularly its native programming language Scala, which for many applications can give much better performance than e.g. Python.

Frameworks like Apache Spark run on clusters with multiple machines, potentially in the 100's or 1000's. To run a job efficiently on such a cluster, it is essential to reduce communication between the machines. In practice this is done by controlling *shuffling*, which happens when all machines write data to all other machines (see image below for a small example with four machines). In addition to reducing the amount of shuffle data, it is also important to ensure that it is evenly distributed between the machines, since otherwise machines could be overwhelmed by receiving a big proportion of the data, stalling or crashing the entire cluster.



Evenly sized k-mer bins

Minimizer orderings influence the length of super-mers, as well as their distribution into different bins.
We previously introduced [3] the Universal Frequency Ordering for minimizers. This ordering helps control the maximum bin size, ensures an even bin size distribution, and also generates long super-mers. These properties greatly aid distributed k-mer processing. The figure shows a comparison of bin sizes generated by various minimizer orderings for the Tara Oceans marine metagenomic dataset. For our ordering **unifreq**, the maximum bin size was reduced by an order of magnitude, and the size distribution is mostly within a tight range, compared with the commonly used ordering **signature**. In experiments, this k-mer ordering was able to reduce memory usage to 1/8 of what the previous best Spark-based k-mer counter needed.

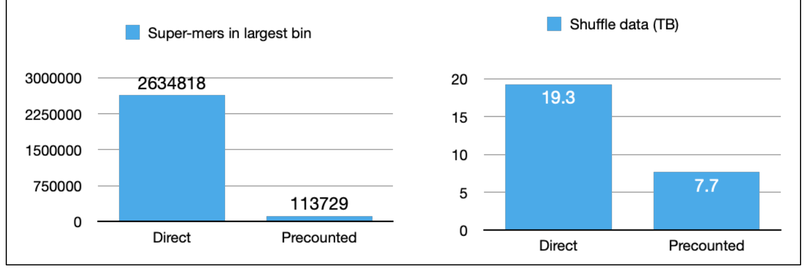


Precounted super-mers

By precounting each super-mer prior to the k-mer counting stage (internally in Spark), we reduce the impact of highly frequent k-mers or patterns in the data, since only distinct super-mers will remain for the final stage. Thus, we can ensure that highly skewed or repetitive data has no impact on the final load balancing. We also reduce the total size of the shuffle data by removing repeated elements.

On the other hand, heterogeneous, non-repetitive data can be load balanced with minimizers, as we have seen (left). Thus, *the combination of pre-counted superkmers and wide minimizers with the universal frequency ordering will be able to load balance any dataset, ensuring good performance.*

The charts show the effect of this approach on the Serratus dataset (below).



Open source

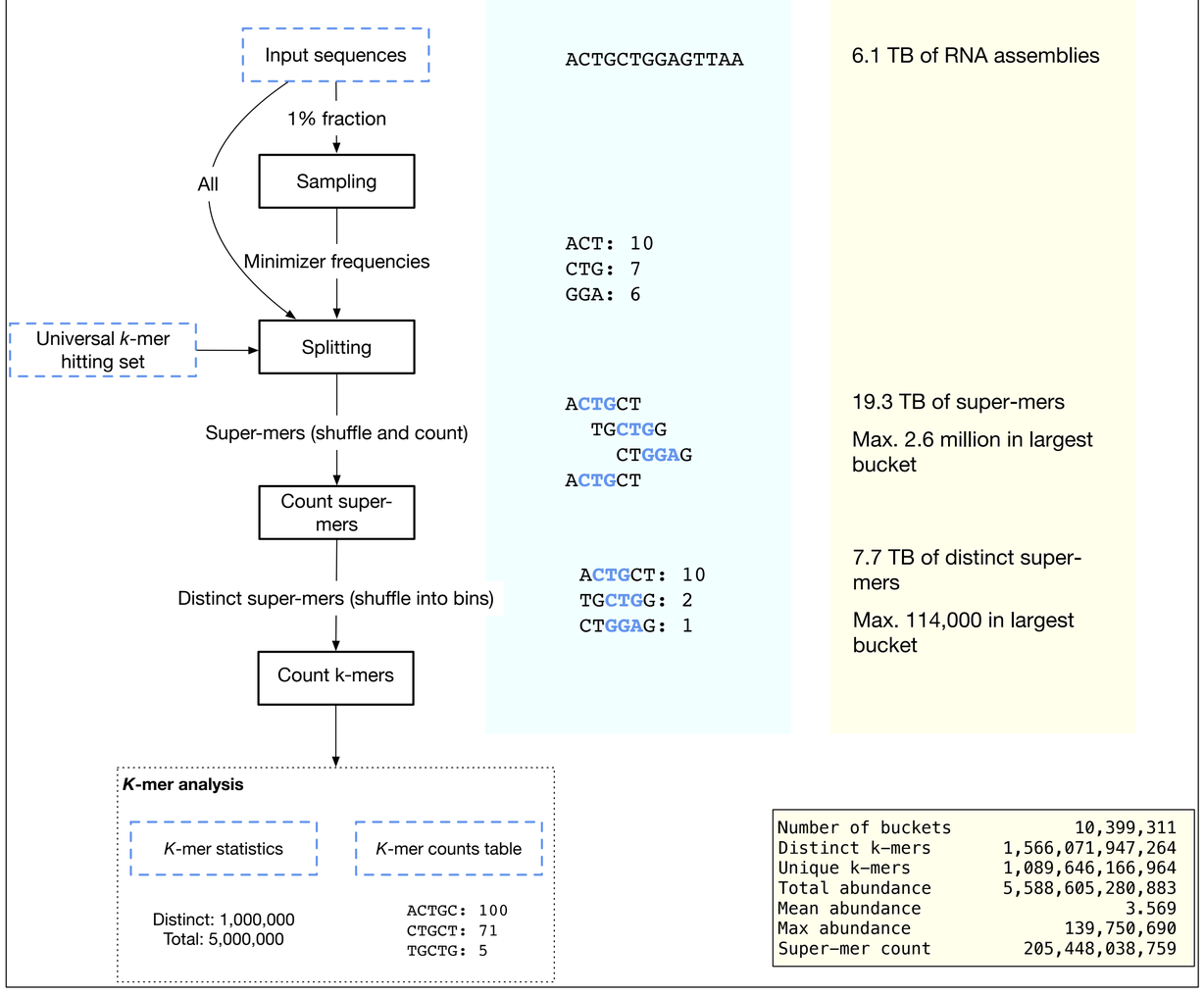
Our k-mer counter "Discount", based on Apache Spark, is available as open source. Please see **https://www.jnpsolutions.io/software/**

Or get it from GitHub at: **https://github.com/jtnystrom/discount**

Discount builds on the Fastdoop library[1] to be able to read FASTA and FASTQ files efficiently. It was inspired by FastKmer[4], and minimizer sets are generated by the PASHA tool[5].

K-mer counting the Serratus dataset

To test the overall scalability of our method, we applied it to the Serratus collection of transcriptomic assemblies [2]. For k=28, it had a total of 5.6 trillion k-mers. This is about 20x larger than what we previously studied in [3]. The figure shows our improved counting pipeline for this project (available as open source in Discount 2.3.0). Pre-counted supermers was essential to making this work with modest memory (4 GB per CPU). The counting took 1760 CPUh. Machines had 16 CPUs, 64 GB RAM and 1 TB SSD each.



Number of buckets	10,399,311
Distinct k-mers	1,566,071,947,264
Unique k-mers	1,089,646,166,964
Total abundance	5,588,605,280,883
Mean abundance	3.569
Max abundance	139,750,690
Super-mer count	205,448,038,759

Towards petabyte scale

Spark is generally able to process petabytes of data, and to scale up to clusters having thousands of machines, so it is reasonable to aim for this level as the next step. 1 PB of input data should correspond to quadrillions of k-mers, assuming metagenomic data similar to the Serratus dataset.

Based on our findings from the Serratus dataset, we estimate that the following parameters would be necessary to k-mer count 1 PB of metagenomic data:

Minimizer width (m): 17 or more
K-mer width (k): Freely chosen >= 25
Input size: 1 PB
Super-mer shuffle data size: 3.2 PB when k = 28
Distinct super-mer data size: 1.3 PB when k = 28
Number of buckets: 2.66 x 10^9
Total abundance (k-mer count): 1 x 10^{15} when k = 28

CPU requirement: 300,000 CPU-hours
Memory requirement: 4 GB per CPU

While these parameters are estimated and would need to be confirmed in practice, this points the way to the possibility of mapping the global k-mer space in its entirety. We expect the limiting factor to be handling the 3.2 PB of shuffle data, for which fast storage and networking would be essential.

References

1. Umberto Ferraro Petrillo, Gianluca Roscigno, Giuseppe Cattaneo, Raffaele Giancarlo, FASTdoop: a versatile and efficient library for the input of FASTA and FASTQ files for MapReduce Hadoop bioinformatics applications, Bioinformatics, Volume 33, Issue 10, 15 May 2017, Pages 1575–1577

2. Edgar, R.C., Taylor, J., Lin, V. et al. Petabase-scale sequence alignment catalyses viral discovery. Nature 602, 142–147 (2022). https://doi.org/10.1038/s41586-021-04332-2

3. Johan Nyström-Persson, Gabriel Keeble-Gagnère, Niamat Zawad, Compact and evenly distributed k-mer binning for genomic sequences, Bioinformatics, Volume 37, Issue 17, 1 September 2021, Pages 2563–2569

4. Ferraro Petrillo U, Sorella M, Cattaneo G, Giancarlo R, Rombo SE. Analyzing big datasets of genomic sequences: fast and scalable collection of k-mer statistics. BMC Bioinformatics. 2019 Apr 18;20(Suppl 4):138. doi: 10.1186/s12859-019-2694-8. PMID: 30999863; PMCID: PMC6471689.

5. Barış Ekim, Bonnie Berger, and Yaron Orenstein. 2020. A Randomized Parallel Algorithm for Efficiently Finding Near-Optimal Universal Hitting Sets. In Research in Computational Molecular Biology: 24th Annual International Conference, RECOMB 2020, Padua, Italy, May 10–13, 2020, Proceedings. Springer-Verlag, Berlin, Heidelberg, 37–53. https://doi.org/10.1007/978-3-030-45257-5_3