Extended abstract

SeqBIM

Correcting Long-Reads with k-mers: A Dream Comes True

Pierre Marijon[1], Philipp Spohr[2] Antoine Limasset[3],

[1]*Heinrich Heine University Düsseldorf Medical Faculty Institute for Medical Biometry and Bioinformatics*
[2]*Heinrich Heine University Düsseldorf, Algorithmic Bioinformatics*
[3]*Univ. Lille, CNRS, UMR 9189 - CRIStAL, F-59000 Lille*
***Corresponding author**: pierre.marijon@hhu.de

Abstract
Long-read sequencing technologies have become widespread for a broad application range. Nevertheless, they still have a high error rate. The correction of such reads is usually time and memory expensive due to the pairwise alignment step. However, previous techniques based on k-mer spectrum analysis performed very well on short-reads and even on long-reads given the availability of short-read sequences. Those techniques were able to be extremely fast and lightweight, relying on a very efficient data structure.

In this presentation, we present two ongoing works showing that such techniques can be adapted to work on noisy long reads directly without relying on precise short read data: `PanCov-Correct` corrects Nanopore reads while keeping low-covered variants, in heterozygous context. `Pcon` & `Br` performs long-read self-correction. These two correction methods reach an error-rate of around 0.1%, faster than other self-correction methods.

Keywords
long-read — correction — variant calling

1. Introduction

Third-generation DNA sequencing is increasingly becoming a standard technology for reference genome construction (de novo assembly), detection of structural variants, long-range variant phasing, and sequencing of GC-rich regions with accurate coverage. However, their higher error rate, which can exceed 10%, and complex error profiles presenting substitutions, deletions, and insertions lead to algorithmic challenges.

Most correction tools use pairwise alignment beforehand to perform the long-read correction. This step is usually the bottleneck in terms of time and memory of most pipelines. We will present preliminary results of two new correction methods based on k-mers from long-reads during this talk. `PanCov-Correct` that corrects reads from COVID-19 while preserving low-covered variants. `Pcon` and `Br` that adapt the well known k-mer spectrum correction to the long-reads error distribution.

2. Materials & Methods

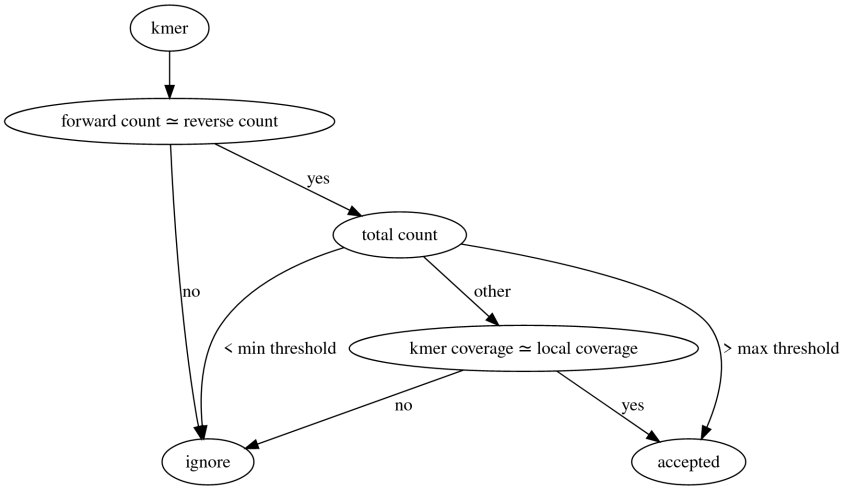

Figure 1. We apply this decision flow to select reads k-mers in the `PanCov-Correct` method i) We only consider k-mers that verify $\min\left(\frac{forwardCount}{totalCount}, \frac{reverseCount}{totalCount}\right) > 0.3$ ii) A k-mer is ignored if its total count is lower than 10 (min threshold) and accepted if upper than 50 (max threshold) iii) For a medium total coverage: we estimate the local coverage of the considered k-mer and accept the k-mer if $kmerCoverage \times 0.7 > estimateLocalCoverage$. All threshold are chosen empirically.

2.1 `PanCov-Correct`

This correction method is part of a larger pipeline `PanCov` (publication in preparation), whose goal is to call variants in COVID-19 samples sequenced with Nanopore technology.

The current state-of-the-art method to call variants of COVID-19 based on Nanopore sequencing, proposed by the ARTIC network [1], can be roughly summarized as (i) reverse transcription from RNA to DNA, (ii) viral genome amplification or enrichment, (iii) generation of the sequencing data (iv) run `Nanopolish` [1] and `Medaka`[2] on this data to produce a consensus sequence and variant calling. This pipeline detects the most abundant viral alleles present in each patient, but exhibits reduced sensitivity for alleles present at lower frequencies and often fails to detect mixed strands in samples.

The sequencing pipeline produces Nanopore reads with a 300 base pairs length, 9% error rate 300x of coverage (with some region's coverage as low as 20x), and some strand bias. Our goal is to provide a correction tool that removes the majority of sequencing errors as well as the strand bias while keeping variants with a low allele frequency.

The correction performed by `PanCov-Correct` is based on `GraphAligner`'s[2] hybrid correction method. `GraphAligner` runs `bcalm` [3] on short-reads to build a *DeBruijn* graph, maps long-reads on this graph, and uses graph information as a ground truth to correct long-reads; this method helps preserving variants if they are

[1] https://artic.network/
[2] https://github.com/nanoporetech/medaka

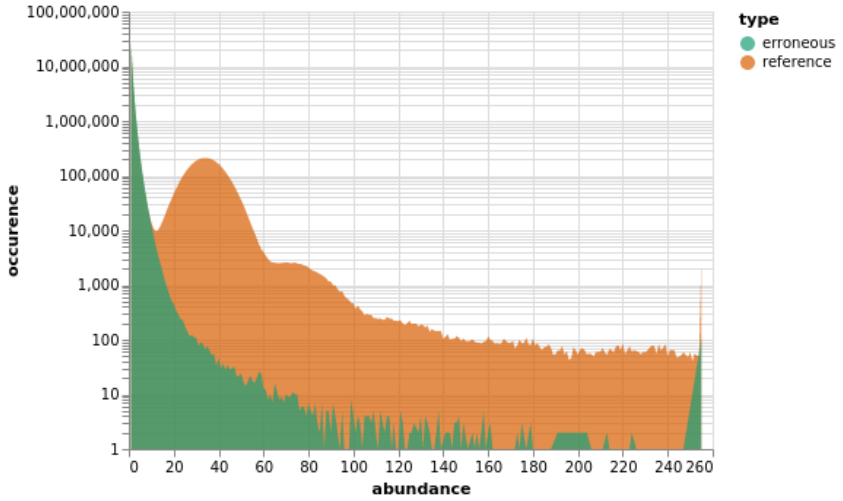

Figure 2. We can see erroneous k-mers generally have a low abundance, the majority of reference k-mers have an abundance of around 40 for this dataset. By replacing low abundance k-mers with high abundant ones we correct the errors.

present in the *DeBruijn* graph.

To replace k-mers from short-reads, we use the reference k-mers, and additional k-mers from Nanopore reads. After a count of k-mers in both strand (with `jellyfish` [4]) we select a subset with the decision tree presented in Figure 1.

The first filter removes erroneous k-mers due to strand bias, and the two other filters erroneous k-mers while keeping k-mers in low coverage regions.

With all accepted k-mers, as well as the k-mers from the reference, we build a *DeBruijn* graph with `bcalm` and use `GraphAligner` to correct reads. We execute this pipeline iteratively with an increasing value for the k-mer size.

2.2 `Pcon` & `Br`

`Pcon` & `Br` correction method is based on the k-mer spectrum and the idea that erroneous k-mers are observed less frequently than correct k-mers in the dataset. Figure 2 shows a k-mer spectrum with reference and erroneous k-mers in 50x *E. coli* Nanopore R10.3 with an error rate of around 5%. We can see that most erroneous k-mers have low coverage.

With `Pcon` and `Br`, we try to apply this method to long-reads. `Pcon` is a k-mer counter designed to count short k-mers quickly. `Br` scans reads and tries to replace low count k-mers, designated as *weak*, by k-mers with high coverage, design now as *solid*, as Musket[5] and Lighter[6].

2.2.1 `Pcon`

`Pcon` is based on a simple reversible hash function that maps all possible canonical k-mer to the range between 0 to $2^{(2\times k)-1}$. By allocating a table of size $2^{(2\times k)-1}$ `Pcon` can store the counts of all possible canonical k-mers.

Each k-mer exists in two versions, forward and reverse, that are considered indistinguishable. Therefore, to reduce memory usage, we store only one version of each k-mer. The chosen form is dubbed canonical. Most of the time, the smallest integer

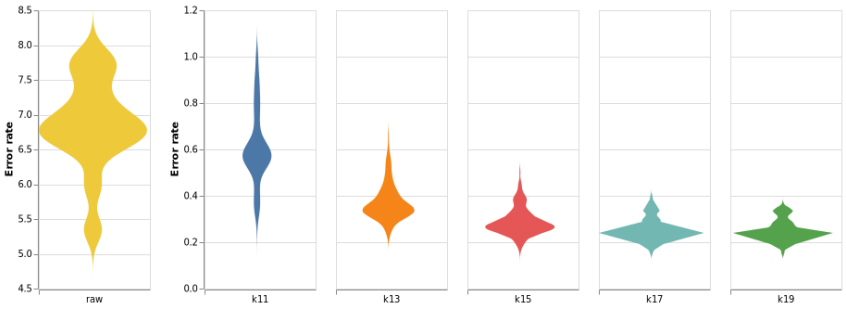

Figure 3. We present the mean error rate of each sample as a violin plot, for the raw reads and after each correction round. The first round with k=11 divides the error rate by more than 10 and further iterations allow the reads to reach an average error of 0.24%. The correction method also reduces the variance of mean error.

value is chosen as the canonical. The `Pcon` hash function is designed to avoid the computation of two versions to get the canonical one.

First, we define a function `popcount`, which returns the number of bits equal to 1 in a k-mer binary representation. The hash function converts the nucleotides of a k-mer in a two bits representation following this encoding: $A \leftarrow 00$, $C \leftarrow 01$, $G \leftarrow 11$ and $T \leftarrow 10$. This encoding's main property is that, given that the k-mer length is odd, `popcount`$(forward)$ is odd iff `popcount`$(reverse)$ is even. Thus, we define the canonical version of a k-mer as the one with even `popcount`. This property helps to determine the canonical k-mer without computing its reverse complement. Moreover, if we work only with canonical k-mers, we can remove one bit for each k-mer and reconstruct this bit by adding a one or a zero to get an even `popcount`.

`Pcon` can convert its count in a bitfield. If a k-mer count is larger than a threshold, the bit corresponding to the k-mer value is set to 1 else to 0.

`Pcon` can write its result in different formats: pcon (a gzip-compressed dump of the count table), CSV, solid (a gzip-compressed dump of bitfield), and it can also produce a k-mer spectrum. `Pcon` is usable as a standalone tool and as a Rust library with C and Python bindings.

2.2.2 `Br`

`Br` screens the read sequence uses `Pcon`'s solid bitfield to know if each k-mer is *solid* or not. When `Br` detects a *weak* k-mer, it can apply four different algorithms to correct the faulty nucleotides. Here we describe those four algorithms dubbed: **One**, **GapLength**, **Graph**, and **Greedy**.

One This algorithm is the simplest one. It supposes that a single isolated error produced the *weak* k-mer. This type of error generates a succession of k *weak* k-mers. This algorithm tries to replace the last base of the first *weak* k-mer to convert it in *solid* k-mer. This base is considered the correct one. If the N following k-mers are *solid* with this correction, `Br` validates this correction. If two or more corrections are possible, `Br` does not try to correct this error.

Graph This algorithm assumes an error generates a succession of *weak* k-mers bordered by two *solid* k-mers. **Graph** algorithm considers the set of *solid* k-mers as

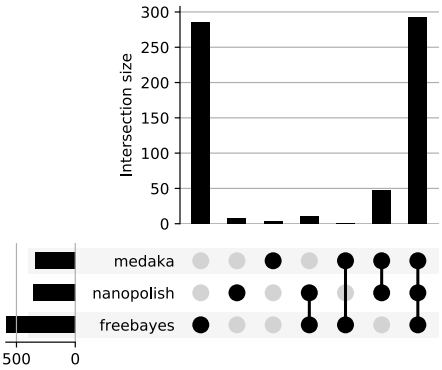

Figure 4. We present the discovered variants with Nanopore reads compared between different methods, `freebayes` corresponds to `freebayes` on the corrected read, `nanopolish` and `medaka` correspond to variants called by the ARTIC network pipeline. We observe that the amount of variant discovered exclusively by freebayes (first bar) is comparable to the amount of variant discovered by all tools (last bar)

a *DeBruijn* graph and searches a simple path between the two border *solid* k-mers. **Graph** starts from the last *solid* k-mer before the error and progresses in the *DeBruijn* graph, if we reach the first *solid* k-mer after error, we replace the erroneous section of the read by the simple path. If we reach a fork, a dead-end, or a cycle during *DeBruijn* graph exploration, `Br` does not apply any correction.

GapLength Inspired by `MindTheGap` [7], the length of *weak* k-mers allows the determination of the error type:

- If $length < k$, it is a deletion
- If $length == k$, it is an error generated by a single nucleotide change
- If $length > k$, it is a substitution or insertion with length equal to the number of *weak* k-mers minus size of k-mer

In practice, we apply the **Graph** algorithm for deletion and the **One** algorithm for isolated errors. For the last case, we can determine the number of k-mers required to replace the last erroneous base. We apply a very similar algorithm as **Graph**, but we can stop the graph exploration earlier.

Greedy Contrary to **Graph** and **GapLength**, **Greedy** does not try to analyze the errors to correct them but directly tries to replace the erroneous sequence by a path in the *DeBruijn*. This algorithm explores the *DeBruijn* graph and tries to find when the graph path matches the read sequence.

3. Result

3.1 PanCov-correct

To evaluate the `PanCov-Correct` method, we run it iteratively with a k-mer size ranging from 11 to 19 and a step size of 2. We evaluate the error rate by mapping

Dataset	Organisme	Technology	Error rate	Coverage
bacteria	*E. coli*	Nanopore R10.0	14.7%	$\approx 127x$
bacteria5	*E. coli*	Nanopore R10.3	5.9%	$\approx 54x$
bacteria7	*E. coli*	Nanopore R10.3	7.7%	$\approx 127x$
celegans	*C. elegans*	Badreads	5 %	16x, 20x, 50x to 400x per 50x step
metagenome	metagenome[3]	Nanopore R10.3	10.8%	
synthetic	*E. coli*	Badreads	1% to 10% per 1% step	$\approx 50x$
yeast	*C. elegans*	Nanopore R10.3	5 %	$\approx 283x$

Table 1. Main characteristic of the data sets used to evaluate `Pcon` & `Br` against other tools.

reads against the reference with `minimap2` and compute the mean error rate with `samtools stats`. The result is presented in Figure 3. By correcting reads with k equals 11, we reduce the error rate by ten. Another iteration improves read quality, just above 0.2% of error. We start with a k-mer size equal to 11 because most *DeBruijn* graphs built from reads are composed of one connected component with this k-mer size.

To evaluate the effect of correction on downstream analysis, we call variants on the corrected reads with `freebayes`. We compare the set of variants found in any dataset between `freebayes`, `nanopolish` and `medaka`. The result is presented in Figure 4. We can notice an important number of variants common with all variant callers, but half of the variants found by `freebayes` on the corrected reads are found exclusively by `freebayes`.

This comparison is not straightforward because we compare variant calling on raw reads and corrected reads. Moreover, we did not have a ground truth like manual curation or variant calling with second-generation reads to evaluate if the variants found only by `freebayes` are genuine variants. However, we plan to do this analysis in the future.

3.2 `Pcon` & `Br`

To evaluate `Pcon` & `Br`, we use some real and synthetic datasets. A resume of the main properties of those datasets are present in Table 1

To evaluate `Pcon`'s performance we compare wall clock time and memory usage against two other tools `jellyfish` [4] and `kmc` [8] (in ram mode), on `metagenomic` dataset reads. The results are presented in Figure 5. We observe that `Pcon` is the tool that has the best wall clock computation and memory usage (except for $k = 19$.

To evaluate `Br`'s performance we computing the error rate with the same method used previously on `PanCov-Correct` and compare our results with other self-correction tools: `CONSENT` [9], `NECAT` [10] and `Canu` [11] correction module. Results on our dataset are shown in Figure 6.

We observe that the runtime of `Br` grows slower than that of other tools. `Br` memory usage with $k = 19$ is large (140 Gb), but constant. The initial error rate impacts the corrected error rate for all tools, but this impact is larger for `Br`. On bacterial datasets with relatively small error rates, around 6%, `Br` performs similarly to other tools. Nevertheless, on more complex datasets like yeast and C. elegans, `Br`] has room for improvement.

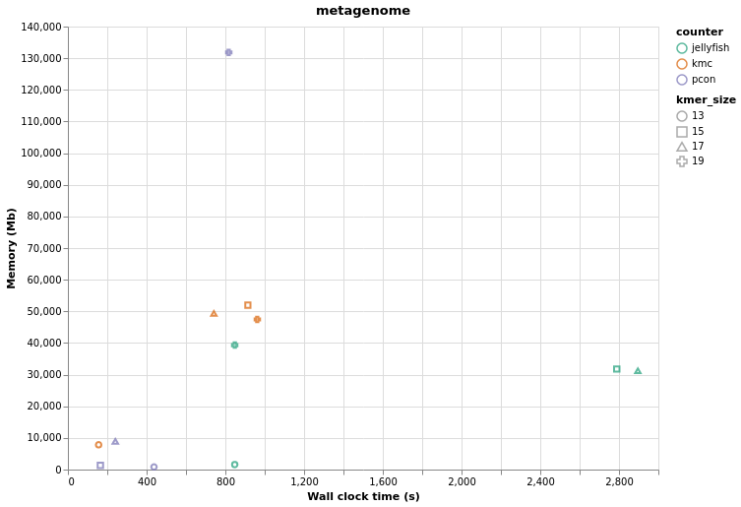

Figure 5. Comparison of `Pcon` computation time and memory usage (in purple) against different k-mers counter on different k-mer size.

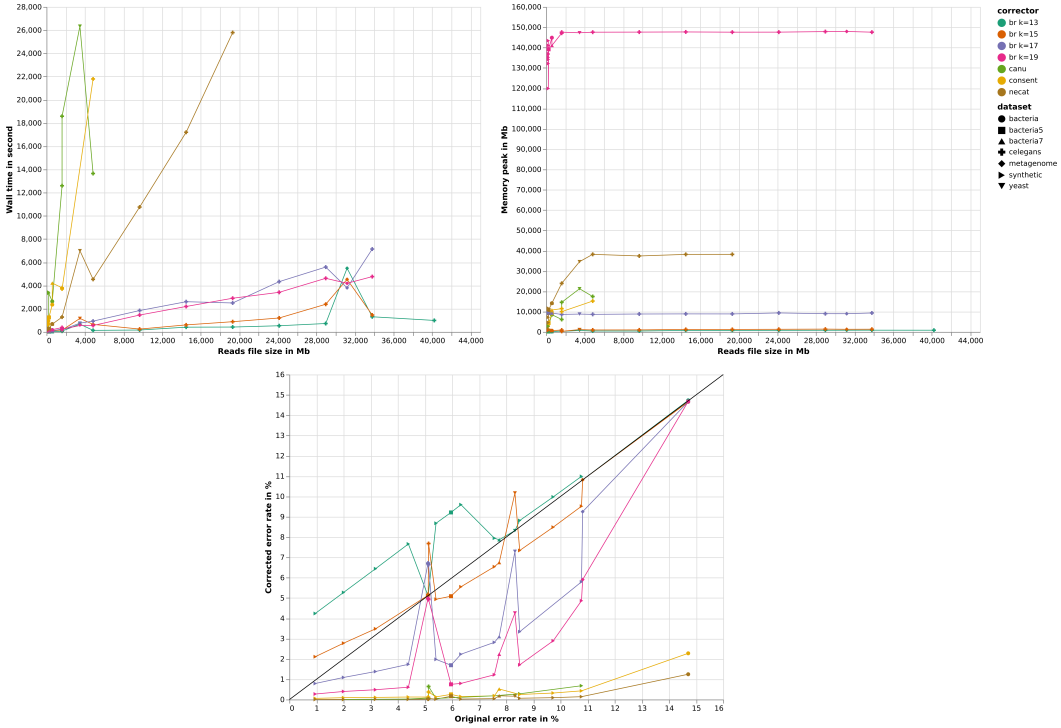

Figure 6. Runtime, memory usage and error rate, of `Br`, `CONSENT`, `NECAT` and `Canu` each point corresponds to a dataset. `Canu`, `CONSENT` and `NECAT` didn't finish in less than 9 hour on all dataset. The black line correspond to identity between original and corrected error rate.

4. Conclusion

`PanCov-Correct` produces good results in terms of correction quality, and downstream analysis allows the discovery of new variants in the COVID-19 sample. We plan to create a standalone tool with a similar method to apply it easily to other organisms.

`Pcon` for its specific target, k-mers with an abundance lower than 20, is faster than other k-mer counters. Preliminary results show that `Br` performs good correction faster than other tools, and recent improvement of raw long-read quality could help `Br` to perform a better correction. Another improvement could be to use another set structure than the one provided by `Pcon`, to reduce the memory impact and use larger k-mer.

These results on two different correction methods demonstrate that k-mer-based correction can be applied to long-read sequences. Our ongoing focus is to improve correction quality, but our first and naive approaches show those strategies' potential.

Acknowledgement

The Centre for Information and Media Technology at Heinrich Heine University Düsseldorf provided computational infrastructure and support.

References

[1] Nicholas J Loman, Joshua Quick, and Jared T Simpson. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature Methods*, 12(8):733–735, jun 2015.

[2] Mikko Rautiainen and Tobias Marschall. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biology*, 21(1), sep 2020.

[3] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, jun 2016.

[4] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, jan 2011.

[5] Yongchao Liu, Jan Schröder, and Bertil Schmidt. Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data. *Bioinformatics*, 29(3):308–315, nov 2012.

[6] Li Song, Liliana Florea, and Ben Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biology*, 15(11), nov 2014.

[7] G. Rizk, A. Gouin, R. Chikhi, and C. Lemaitre. MindTheGap: integrated detection and assembly of short and long insertions. *Bioinformatics*, 30(24):3451–3457, aug 2014.

[8] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, may 2017.

[9] Pierre Morisse, Camille Marchet, Antoine Limasset, Thierry Lecroq, and Arnaud Lefebvre. CONSENT: Scalable long read self-correction and assembly polishing with multiple sequence alignment. feb 2019.

[10] Chuan-Le Xiao, Ying Chen, Shang-Qian Xie, Kai-Ning Chen, Yan Wang, Yue Han, Feng Luo, and Zhi Xie. MECAT: fast mapping, error correction, and de novo assembly for single-molecule sequencing reads. *Nature Methods*, 14(11):1072–1074, sep 2017.

[11] Sergey Koren, Brian P. Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptivek-mer weighting and repeat separation. *Genome Research*, 27(5):722–736, mar 2017.