Bioinformatics, 36(5), 2020, 1374–1381
doi: 10.1093/bioinformatics/btz102
Advance Access Publication Date: 20 February 2019
Original Paper

Sequence analysis

Toward perfect reads: self-correction of short reads via mapping on de Bruijn graphs

Antoine Limasset [1,*], Jean-François Flot [1,2,†] and Pierre Peterlongo [3,†]

[1]Evolutionary Biology & Ecology, Université Libre de Bruxelles (ULB), Bruxelles, Belgium, [2]Interuniversity Institute of Bioinformatics in Brussels – (IB)[2], Brussels, Belgium and [3]Inria, CNRS, University of Rennes, IRISA, Rennes, France

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the last two authors should be regarded as Joint Last Authors.

Associate Editor: Alfonso Valencia

Received on July 23, 2018; revised on January 7, 2019; editorial decision on February 7, 2019; accepted on February 18, 2019

Abstract

Motivation: Short-read accuracy is important for downstream analyses such as genome assembly and hybrid long-read correction. Despite much work on short-read correction, present-day correctors either do not scale well on large datasets or consider reads as mere suites of *k*-mers, without taking into account their full-length sequence information.

Results: We propose a new method to correct short reads using de Bruijn graphs and implement it as a tool called Bcool. As a first step, Bcool constructs a compacted de Bruijn graph from the reads. This graph is filtered on the basis of *k*-mer abundance then of unitig abundance, thereby removing most sequencing errors. The cleaned graph is then used as a reference on which the reads are mapped to correct them. We show that this approach yields more accurate reads than *k*-mer-spectrum correctors while being scalable to human-size genomic datasets and beyond.

Availability and implementation: The implementation is open source, available at http://github.com/Malfoy/BCOOL under the Affero GPL license and as a Bioconda package.

Contact: antoine.limasset@gmail.com

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

1.1 Why correct reads?

Genome sequencing is a fast-changing field. Two decades have seen three generations of sequencing technologies: Sanger electropherograms (a.k.a. first-generation sequencing), short reads from second-generation sequencing (SGS) and long, error-prone reads from third-generation sequencing (TGS). Albeit powerful, these technologies all come with stochastic errors, and for some, non-stochastic ones. Stochastic errors are usually corrected using a consensus approach leveraging the high coverage depth available in most genome projects, whereas non-stochastic errors can be eliminated by 'polishing' the sequences, generally post-assembly, using reads obtained from a different sequencing technique and/or sophisticated error models (Loman *et al.*, 2015).

Most *de novo* assemblers following the overlap–layout–consensus (OLC) paradigm handle the stochastic errors present in the reads during the consensus step toward the end of the assembly process. In contrast, tools following the de Bruijn graph (DBG) paradigm generally attempt to filter out erroneous *k*-mers by considering only *k*-mers present at least a minimal number of times in the reads to be assembled. Both paradigms may benefit from a preliminary error-correction step. In the case of DBG assemblers, lowering the error rate in the reads to be assembled makes it possible to use a larger *k*-mer size, paving the way for a more contiguous assembly. With OLC assemblers, a lower error rate allows more stringent alignment parameters to be used, thereby improving the speed of the process and reducing the amount of spurious overlaps detected between reads.

Beyond *de novo* assembly, other applications that require mapping, such as SNP calling, genotyping or taxonomic assignation, may also benefit from a preliminary error-correction step aimed at increasing the signal/noise ratio and/or reducing the computational cost of detecting errors *a posteriori* (DePristo *et al.*, 2011; Li *et al.*, 2009).

1.2 On the use of short reads as long reads are rising

Although long-read technologies from TGS, marketed by Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), are on the rise for many purposes such as genome assembly [as TGS reads are order of magnitude longer and TGS assemblers are therefore less sensitive to repeats (Nagarajan and Pop, 2009)], there are reasons to think that SGS will still be broadly used in the next decade. This is because SGS reads remain considerably cheaper and their accurate sequences make them highly valuable when haplotypes or highly similar paralogs are to be distinguished. Besides, recent methods are able to deliver long-distance information based on short-read sequencing. Chromosome conformation capture (Dekker *et al.*, 2002; Flot *et al.*, 2015) provides pairs of reads that have a high probability of originating from the same chromosome, whereas Chromium 10× (Kitzman, 2016) uses a droplet mechanism to ensure that a pack of reads comes from a single DNA fragment of up to hundreds of thousands base pairs. Both of these techniques have been shown to produce assembly continuity comparable to TGS assemblies (Marie-Nelly *et al.*, 2014; Schwager *et al.*, 2017; Yeo *et al.*, 2017). SGS reads are also used jointly with long reads to compensate the latter's high error rate [and systematic homopolymer errors in the case of ONT reads (Jain *et al.*, 2016)] in a cost-efficient way (Wick *et al.*, 2017). These different applications make it worth investing effort in improving short-read correctors beyond the current state of the art, in the hope that near-perfect reads will positively impact all downstream analyses that require accurate sequences.

1.3 *k*-mer-spectrum techniques

k-mer-spectrum techniques are conceptually the simplest correction method and remain broadly used. The underlying intuition is that true genomic *k*-mers will be seen many times in the read set, whereas erroneous *k*-mers originating from sequencing errors will be comparatively much rarer. The first step to correct reads using this approach is therefore to choose an abundance threshold [above which *k*-mers are called 'solid' and below which they are called 'weak' (Pevzner *et al.*, 2001)]. *k*-mer-spectrum correctors aim to detect all weak *k*-mers in the reads and correct them by turning them into solid ones.

One of the best *k*-mer-spectrum correctors available to date (Akogwu *et al.*, 2016) is Musket (Liu *et al.*, 2013); however, its memory consumption is high on large genomes because of its indexing structure. Another tool, Bloocoo (Benoit *et al.*, 2014), achieves a comparatively lower memory footprint, even on genomes comprising billions of bases (such as the human one), by using a Bloom filter to index *k*-mers. Lighter (Song *et al.*, 2014) also uses Bloom filters but bypasses the *k*-mer-counting phase by only looking at a subset of the *k*-mers in a given dataset, therefore achieving greater speed. One problem with these tools is that they rely on local *k*-mer composition with a bounded *k*-mer size ($<= 32$), which limits correction power on complex datasets. A less 'local' approach is implemented in the BFC (Li, 2015) corrector, which attempts to correct each read as a whole by finding the minimal number of substitutions required for a read to be entirely covered by solid *k*-mers.

1.4 Other read correction techniques

Other correction techniques rely either on suffix arrays [allowing the use of substrings of various sizes instead of only fixed *k*-length words (Schröder *et al.*, 2009)] or on multiple-read alignments (Salmela and Schröder, 2011). Despite their methodological appeal, these techniques are resource-expensive and do not scale well on large datasets. Moreover, benchmarks suggest that they perform significantly worse than state-of-the-art *k*-mer-spectrum correctors (Yang *et al.*, 2013). Another approach for correcting reads, pioneered by LoRDEC (Salmela and Rivals, 2014) then by LoRMA (Salmela *et al.*, 2017), is to use DBGs instead of strings as a basis for correction. In the LoRDEC approach, this DBG is built from highly accurate short reads and used to correct long reads. In LoRMA, the DBG is built from the very same long reads that the program is attempting to correct. Both of these tools are geared toward correcting long reads, but another program, Rcorrector (Song and Florea, 2015), uses abundance-annotated DBGs to correct Illumina reads with a focus on RNA-seq data. Still, these three DBG-based approaches use comparatively small *k*-mer sizes (by default 19 for LoRDEC and for Rcorrector), making them likely to confound errors in repeated regions.

Here, we implement a DBG-based corrector that uses large *k*-mer sizes (up to the length of the reads). We then show that this corrector, dubbed Bcool, vastly outperforms state-of-the-art *k*-mer-spectrum correctors and existing DBG-based approaches while being both scalable and resource-efficient.

2 Materials and methods

The intuition behind *k*-mer-spectrum correction is that *k*-mers, once filtered according to their abundance, represent a reference that can be used to correct reads. The idea that a DBG provides a better reference than a *k*-mer set might be surprising at first glance since a DBG is equivalent to its set of *k*-mers. However, a novelty of our approach is that we build a compacted DBG, that is, a DBG in which non-branching paths are turned into unitigs. We are therefore able to remove erroneous *k*-mers by relying on the topology of the graph rather than on their abundance alone. This results in a better distinction between erroneous and genomic *k*-mers. The second novelty is to perform whole-read alignments on this cleaned graph, thereby achieving a global correction of each read and allowing the use of high *k*-mer sizes for improved correction of repetitive and/or large genomes.

After briefly reviewing several limitations of *k*-mer-spectrum correction (Section 2.1), we describe the key parts of our workflow while detailing how our proposed approach tackles these issues (Section 2.2).

2.1 *k*-mer-spectrum limitations

In this section we identify four sources of miscorrection in *k*-mer-spectrum approaches, as represented in Figure 1. As mentioned previously, *k*-mer-spectrum correctors infer a set of solid *k*-mers that are used to correct reads. Erroneous *k*-mers (i.e. *k*-mers containing at least one sequencing error) can be filtered out by keeping only *k*-mers above a given abundance threshold, called the solidity threshold. As mentioned earlier, *k*-mers with an abundance higher than or equal to this threshold are called 'solid', whereas *k*-mers that are less abundant are called 'weak'.

2.1.1 Weak genomic *k*-mers

Depending on the solidity threshold chosen, random variations in sequencing depth may cause some genomic *k*-mers to fall below the threshold and be erroneously filtered out. This kind of *k*-mer creates

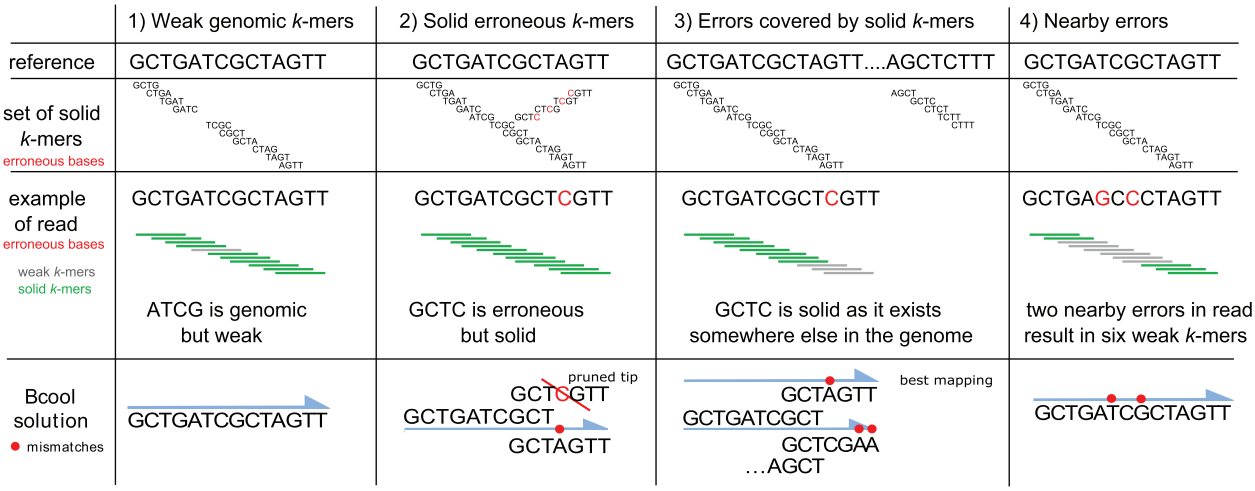

Fig. 1. Four issues with k-mer-spectrum methods, and how Bcool handles them (blue half-arrows represent the paths of the graph on which given reads map). (1) Genomic *k*-mers may be appear weak because of their low abundance: by using a very low *k*-mer abundance threshold coupled with a unitig abundance threshold, Bcool retains low-abundance *k*-mers and manages to correct the reads that contain them. (2) Erroneous *k*-mers may appear solid because of their high abundance: Bcool detects the tip pattern produced by such solid erroneous *k*-mers and is therefore able to discard them (other erroneous *k*-mers are detected at the unitig filtering step). (3) Sequencing errors may be validated by genomic *k*-mers originating from other parts of the genome: by considering mappings globally, Bcool chooses the best path for each read, i.e. the one on which it maps with the smallest number of mismatches. (4) Multiple errors may occur on a *k*-mer, resulting in a large weak region: by using unitigs instead of *k*-mers to correct reads, Bwise is able to correct properly reads that contain several nearby errors

a situation such as the one represented in Figure 1.1, where an isolated weak *k*-mer is flagged as a putative error on a read that is actually correct. Since at least *k* successive weak *k*-mers are expected to be seen when there is a sequencing error, *k*-mer-spectrum-based correctors normally consider isolated weak *k*-mers as likely to be simply missed genomic *k*-mers and do not attempt to correct them.

2.1.2 Solid erroneous *k*-mers

Conversely, setting the solidity threshold too low may lead to the inclusion of some erroneous *k*-mers in the trusted set of solid *k*-mers (Fig. 1.2). As a result, the errors in the reads harboring such *k*-mers are not corrected and may also propagate to other reads if these *k*-mers are used for correction.

2.1.3 Errors covered by solid *k*-mers

If a sequencing error in a read is covered partly or entirely by genomic *k*-mers originating from other regions of the genome, the error may not be detected and it may be difficult to correct it as the remaining isolated weak *k*-mers will be hard to distinguish from the pattern in point 1 above. Such situations are likely to occur in repeated or quasi-repeated genome regions, leading to complex situations where genomic sequences may have several contexts.

2.1.4 Nearby errors

Accurate correction is also complex when multiple errors occur nearby each other (i.e. less than *k* bases apart). In such situations, the number of errors and their positions cannot be easily estimated, and *k*-mer-spectrum correctors have to perform a very large number of queries to correct them. Musket uses an aggressive greedy heuristic that tries to replace the first weak *k*-mer encountered by a solid one then checks whether the next *k*-mers became solid as a result. But, as shown in Figure 1.4, this heuristic is inefficient if the *k*-mers that follow contain other sequencing errors.

2.1.5 *k*-mer size

All the issues highlighted above boil down to a central problem when using *k*-mer-spectrum correctors: the size of *k*. If a too large *k*-mer size is used, most *k*-mers contain at least one sequencing error and many of them actually contain several errors. In those cases, *k*-mer-spectrum correctors may be unable to locate errors and to perform correction as they rely on genomic *k*-mers to find possible candidates to replace weak ones. On the contrary, if *k* is too small, most *k*-mers are solid and almost no correction is performed. As *k*-mer-spectrum correctors are usually geared toward correcting SGS reads, they use a *k*-mer size around 31 that is well suited for the error rate of Illumina data. Choosing a larger *k* results in suboptimal correction or even in a failure of the program to run (see Supplementary Materials). This limitation may be a problem when addressing large and repeat-rich genomes, as a large number of *k*-mers are repeated in various contexts throughout the genomes and large *k*-values are required to distinguish them.

2.2 DBG-based read correction

In this section, we describe our proposal, called Bcool (which stands for 'de Bruijn graph-based read correction by alignment'). The basic idea is to construct a DBG from the read set, to clean it, and then to map the reads on the DBG. Reads that map with less than a threshold number of mismatches are corrected using the graph sequence, which is supposed to be almost error-free. An important feature of Bcool is that the graph is constructed by filtering out low-coverage *k*-mers and additionally by discarding unitigs according to topological information (see Section 2.2.2), yielding an almost perfect reference graph.

We present in Figure 1 some simple examples illustrating how our DBG-based read correction handles the problems listed above. Our proposal differs from *k*-mer-spectrum techniques in that our reference is a cleaned compacted DBG (Chikhi *et al.*, 2015) instead of a set of *k*-mers, and that we map the reads onto the graph instead of looking at all *k*-mers contained in the reads. With high-coverage data (typically above 50× coverage depth), our approach also allows the use of large *k*-mer sizes up to read length, thereby improving the correction of reads originating from complex, repeated genome regions.

Bcool's workflow is depicted in Figure 2. Each of its components is either an independent tool already published or an independent

Fig. 2. Bcool workflow. The white boxes are FASTA files and the grey boxes represent the tools that process or generate them. ntCard (Mohamadi *et al.*, 2017) is used to select the best-suited *k*-mer size. A compacted DBG is then constructed using Bcalm2 (Chikhi *et al.*, 2016). The Btrim module cleans the graph, and the reads are finally mapped back on the DBG using Bgreat2

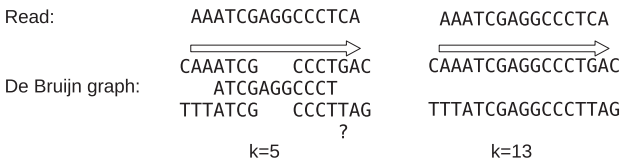

Fig. 3. Using a large *k*-mer size simplifies the graph by lowering the amount of unsolved repeats. In this example a repeat of size 10 is present in the genome in different contexts. With $k = 5$, we are not able to correct the penultimate nucleotide of the read represented by an arrow. But with $k = 13$ we have determined the context of the repeat and know that only two possible paths exist. This way we are able to safely correct the read

module that could be reused in other frameworks. We describe below the different key steps of the workflow.

2.2.1 *k*-mer size selection

We aim to use the highest possible *k*-value to resolve as many repeats as possible (up to *k* length), thereby improving the correction as shown in Figure 3. However, choosing a *k*-value too large would yield a fragmented graph. Therefore, we implemented an automated tool, somewhat similar to *k*-merGenie (Chikhi and Medvedev, 2014), that uses ntCard to estimate the *k*-mer spectrum of the dataset for several values of *k*. Our approach detects the first local minimum for each *k*-mer spectrum then selects the highest *k*-value for which this minimum is above the unitig threshold. This way, we expect to keep most genomic *k*-mers that are more abundant than the unitig threshold. This approach is more conservative and simpler than the one implemented in *k*-merGenie, which attempts to fit the *k*-mer spectrum on a haploid or diploid model with the aim of finding the *k*-value most suitable for assembling the reads.

2.2.2 DBG construction and cleaning

In Bcool, the DBG is constructed using Bcalm2 (Chikhi *et al.*, 2016), a resource-efficient method to build a compacted DBG. In a compacted DBG, nodes are not composed of single *k*-mers but of unitigs (i.e. maximal simple paths of the DBG) of lengths larger than or equal to *k*.

A novelty of this work, made possible by the compacted graph representation, is the way we clean up the reference DBG before using it to correct reads. In this context we propose and use the notion of *unitig abundance*, defined as the mean abundance of all the constituent *k*-mers of a given unitig.

The DBG is initially constructed with a very low abundance threshold (by default 2, i.e. *k*-mers that occur only once are considered as probable errors and discarded). This very low threshold value decreases the probability of missing genomic *k*-mers, but as a consequence, many erroneous *k*-mers are not filtered out. A first step to deal with those is to remove short dead-ends (a.k.a. 'tips') from the graph. Formally, we define a tip as a unitig of length inferior to $2 * (k - 1)$ that has no successor at one of its extremities. Such dead-ends mainly result from sequencing errors occurring on the first or last *k* nucleotides of a read.

By contrast, errors located at least *k* nucleotides away from the start or the end of a read form bubble-like patterns: in a second step, we tackle these remaining erroneous *k*-mers by taking a look at unitig abundance. We choose a unitig abundance threshold (higher than the *k*-mer abundance threshold used previously) and when a unitig has an abundance lower than this threshold, we discard it completely. Intuitively, averaging the abundance across each unitig makes it possible to 'rescue' genomic *k*-mers with low abundance (that tend to be lost by *k*-mer-spectrum techniques, see Fig. 1.1) by detecting that they belong to high-abundance unitigs, and these genomic *k*-mers can then be used for correction. Conversely, erroneous *k*-mers are likely to belong to low-abundance unitigs that are filtered out. This unitig abundance threshold can be user-specified or can be selected automatically by looking at the unitig abundance distribution and choosing the first local minimum.

These two cleaning steps are applied several times in an iterative manner to handle complex scenarios where error patterns are nested. This strategy allows us to address the issues depicted in Figures 1.1 and 2. This DBG-cleaning strategy is implemented in a tool named Btrim (https://github.com/Malfoy/BTRIM). Compared with a strategy based only on *k*-mer abundance, our approach keeps more low-abundance genomic *k*-mers while removing more erroneous *k*-mers. As shown in Figure 4, the filtering strategy used by Bcool produces a graph with less erroneous (false positive) *k*-mers and less missed genomic (false negative) *k*-mers than the sole *k*-mer-abundance threshold used by *k*-mer-spectrum correctors, resulting in a better set of *k*-mers. Detailed evaluation of the tipping and unitig-filtering strategies is provided in the Supplementary Materials.

2.2.3 Read mapping

In contrast to *k*-mer-spectrum-based techniques, Bcool uses an explicit representation of the DBG. Although in its current implementation this entails a higher memory usage and computational cost than *k*-mer-spectrum correctors, doing so provides an efficient way to fix the issues depicted in Figures 1.3 and 4. Each read is aligned in full length on the graph, and the correction is made on the basis of the most parsimonious path on which the read maps in the graph.

For mapping reads on the DBG, we use an improved, yet unpublished, version of Bgreat (Limasset *et al.*, 2016) called 'Bgreat2' (https://github.com/Malfoy/BGREAT2). The description of this method is out of the scope of this paper but we provide here the main characteristics of the new implementation. The main improvements are that Bgreat2 has no third-party dependencies (in contrary to the published version) and that it outputs the optimal alignment of each read (instead of returning its first valid alignment). The alignment procedure uses a classical seed-and-extend process. To achieve good mapping performances and to avoid overcorrection of poorly mapped reads, we limit the amount of mismatches allowed according to a parameter (10 by default). Using the graph, the extension phase maps a read on several potential paths, and among all valid alignments, only those minimizing the number of mismatches are considered, using a 'best-first' approach similar to the BFC search. If several different optimal alignments exist, by default the read is not mapped. This choice can optionally be modified to output one of the optimal mappings.

We highlight the fact that knowing the graph structure from the unitig set allows efficient graph exploration. Graph traversal based on a *k*-mer set would require to query the existence of all possible

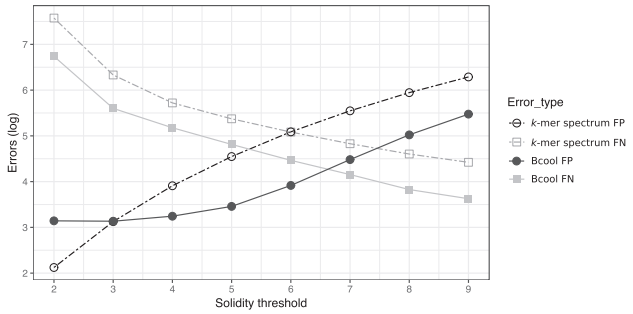

Fig. 4. Impact of the solidity threshold on *k*-mer correction according to the strategy used. The vertical axis shows the number of errors (in a log scale). Errors are either erroneous *k*-mers that are retained in the *k*-mer set (false positives FP, represented with circles) or genomic *k*-mers that are discarded (false negatives FN, represented with squares). In this experiment, we used $k = 63$ on a 50× coverage of 150-bp reads simulated from the *C.elegans* reference genome. The errors produced by Bcool are represented with full lines while those produced by *k*-mer-spectrum correctors are represented with dashed lines. For *k*-mer-spectrum techniques, the solidity threshold applies to *k*-mers, whereas for Bcool it applies to the unitigs constructed from non-unique *k*-mers. We observe that the proposed graph-cleaning operations are able to retain more genomic *k*-mers and remove more erroneous *k*-mers than the raw *k*-mers abundance threshold

successors of every *k*-mer, whereas our proposal needs only to query the index when the end of an unitig is reached. To keep a low memory usage, Bgreat2 uses a minimal perfect hash function (Limasset *et al.*, 2017) for indexing seeds. Moreover, it is possible to subsample seeds for indexation. For instance, by indexing only one out of five seeds, we were able to run Bgreat2 on a human dataset using around 20 GB of RAM at the price of only a slight decrease in correction efficiency (see Table 1).

3 Results

We present results based on simulated datasets as well as on real ones. Simulations make it possible to precisely evaluate correction metrics (Section 3.2) and to assess their impact on downstream assembly (Section 3.3). Correction evaluation was performed using simulated reads from several reference genomes: *Caenorhabditis elegans*, the human chromosome 1, and the whole human genome. In contrast, the results presented in Section 3.4 aim to validate our approach using real data. All experiments were performed on a cluster node with a Xeon E5 2.8 GHz 24-core CPU, 256 GB of memory and a mechanical hard drive.

In what follows, false negatives (FN) stand for non-corrected errors, whereas false positives (FP) are erroneous corrections and true positives (TP) are errors that were correctly corrected. The *correction ratio* is defined as $= (TP + FN)/(FN + FP)$; it is the ratio of the number of errors prior to correction (TP + FN) versus after correction (FN + FP). The higher the correction ratio, the more efficient the tool.

We compared the results obtained using Bcool with several state-of-the-art short-read correctors: Bloocoo (Benoit *et al.*, 2014), Musket (Liu *et al.*, 2013), BFC (Li, 2015) and Lighter (Song *et al.*, 2014). We also tested LoRDEC and Rcorrector (since these correctors rest on a principle similar to Bcool) but they performed rather poorly on our datasets. In the case of LoRDEC the correction ratio obtained was below 10, which may have been expected since this program was designed with long reads in mind and had never been tested on Illumina reads before. In the case of Rcorrector, we could only run it on our *C.elegans* dataset and the correction ratio obtained was about 10. This poor scalability can be explained by the memory-expensive data structure used by this corrector geared toward RNA-seq data, whereas the low correction ratio probably results from its conservative stance toward low-coverage *k*-mers.

3.1 Performance benchmark

Before presenting qualitative results, we first compare the performance of the correctors included in our benchmark. We evaluated the resources used by the different correctors on datasets simulated from the *C.elegans* and human genomes. We present here the memory used, the wall-clock time and the CPU time reported by the Unix *time* command. Our results, presented in Table 1, show that that Bcool has a higher RAM footprint and is slower than the other tools we tested, except Musket. This is due to Bcool's explicit graph representation and its indexation scheme. However, Bcool scales well with genome size, as shown in Table 1. Moreover, it is possible to reduce the memory footprint by sub-sampling during indexation the seeds used for read mapping. This results in a greatly reduced memory footprint at the price of a slight decrease in correction performance. In the human experiment, graph creation took about 8h30 and read mapping took about 3h30. As discussed below, there is clearly room for performance improvements both during the graph-construction phase and the read-mapping phase.

3.2 Correction ratios on simulated datasets

Our results on simulated haploid data are presented in Figure 5. They show that Bcool obtained a correction ratio an order of magnitude above the other tested tools. Note that as shown in Supplementary Materials we tested several other conditions (longer reads, lower coverage, lower error rate), all leading to the same conclusion. In each of our experiments, Bcool had a better correction ratio together with a better precision and recall. The correction precision is critical, as a corrector should not introduce new errors. Our experiments showed a good precision for all the tools we tested (even on a human genome), with a net advantage for Bcool though. For instance, our human-genome experiment with 100× coverage of 150-bp reads yielded a precision of 95.33% for Bloocoo, 96.74% for Lighter and 99.61% for Bcool.

3.2.1 Diploid correction

To assess the impact of heterozygosity, we tested these correctors on simulated human diploid genomes. To obtain a realistic distribution of SNPs and genotypes, we used SNPs predicted from real human individuals and included them in the reference genome. Our results show that, on these datasets, the result quality of all tested correctors remains almost identical to those obtained on haploid simulations. The details and results of this experiment are presented in the Supplementary Material.

3.3 Impact of the correction on assembly

To evaluate the impact of the correction on assembly, we ran the Minia assembler (Chikhi and Rizk, 2013) on uncorrected reads and on read sets corrected with each of the correctors included in our benchmark. For each assembly, we tested several *k*-values (the main parameter of Minia) from $k = 21$ to $k = 141$ with a step of 10. For each corrector, only the best result is presented here. These results, presented in Figure 6, show that the N50 assembly metric is better on data corrected by Bcool. This can be explained by the fact that with a better read correction, a higher *k*-value can be selected, leading to a more contiguous assembly. As an example, for the *C.elegans* genome with 250-bp reads the best *k*-mer size was 91 for the raw reads, 131

Table 1. Performance comparison on *C.elegans* and human simulated 250-bp reads with 1% error rate and 100× coverage using 20 cores

Corrector	RAM used (GB)	Wall-clock time (h:min)	CPU time (h:min)	Correction ratio
C.elegans				
Bloocoo	5.462	0:07	2:02	30.28
Lighter	**0.627**	**0:06**	**1:40**	31.16
Musket	24.755	0:56	16:44	90.33
BFC	8.390	0:13	4:29	14.58
Bcool	12.449	0:21	4:25	**183.53**
Human				
Bloocoo	**10.500**	9:31	90:10	6.79
Lighter	14.121	**4:22**	**60:06**	5.65
Bcool	48.081	11:41	129:39	**23.84**
Bcool -i 5	19.724	15:56	194:41	22.621

Note: Bcool was run with a fixed k-mer size, $k = 63$. Bcool -i 5 indexed only one out of every five seeds to reduce memory usage. BFC and Musket were not able to correct the full human datasets. The best results among correctors are in bold.

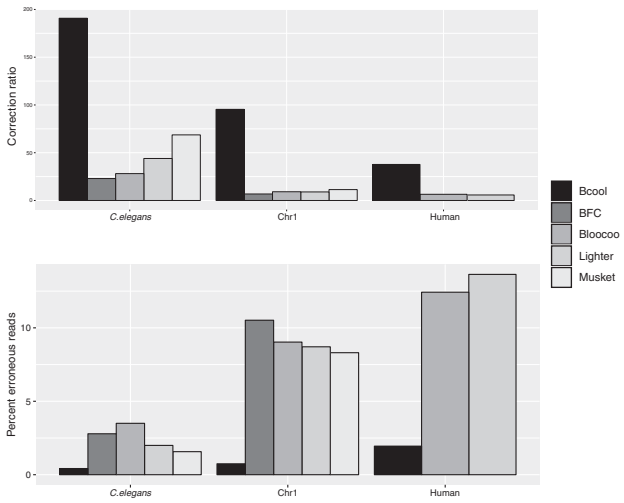

Fig. 5. Correction ratios (top) and percentage of erroneous reads after correction (bottom) for different correctors on our three simulated haploid datasets. We simulated 100× of 150-bp reads with a 1% error rate. BFC and Musket ran out of memory on all full human datasets

for reads corrected using Lighter or Musket, 141 for reads corrected using Bloocoo and 171 for reads corrected using Bcool.

3.4 Real datasets

In this section, we evaluate the impact of the correction on assembly continuity using several real datasets: a *C.elegans* Illumina HiSeq 2500 run with 79.8 million reads of length 150 bp amounting to 12 Gb (DRR050374) (≈120× coverage); and a *Arabidopsis thaliana* Illumina MiSeq run with 33.6 million reads of length 250 bp amounting to 8.4 Gb (ERR2173372) (≈60× coverage). For this benchmark we used the string graph assembler fermi (Li, 2012) given its efficiency and robustness. Assembly reports provided by Quast (Gurevich *et al.*, 2013) are presented in Tables 2 and 3 using contigs larger than 1000 nucleotides. We see that on both datasets Bcool obtains the most contiguous assembly in terms of N50 as well as N75 statistics.

4 Perspective

4.1 Perspectives regarding short reads

We have shown how to construct and clean a reference graph that can be used to efficiently correct sequencing errors. This approach is not to be compared with multiple-k assembly as here we only apply a conservative correction to the graph without trying to remove variants nor to apply heuristics to improve the graph continuity: only k-mers that are very likely to be erroneous are removed in this process. Such conservative modifications on an intermediate graph used as a reference appears a promising approach to better exploit the high accuracy of short reads. The use of a high k-mer size is critical to address the correction problem on large, repeat-rich genomes, and the impossibility for k-mer-spectrum correctors to use a large k-mer size is a major limitations of such approaches. By contrast, our DBG-based solution uses a large k-mer size and therefore yields a more efficient correction on large, repeat-rich genomes. The resulting error-corrected reads are nearly perfect and can be assembled using an overlap-graph algorithm or may be used for other applications, such as variant calling.

Several propositions can be made to further improve Bcool. The read-mapping step could make use of the quality values available in FASTQ files or provide other types of correction, such as read trimming. Adding the capacity to detect and correct indels during the mapping step could allow Bcool to correct other types of sequencing data, such as Ion torrent or PacBio CCS reads. The performance of the pipeline could also be globally improved. The DBG construction could implement techniques similar to the sub-sampling used by Lighter to reduce its disk usage and therefore improve its running time. Besides, our mapping method is still naive, and implementing efficient heuristics such as the ones used by BWA (Li and Durbin, 2009) and Bowtie2 (Langmead and Salzberg, 2012) could greatly improve the throughput of Bcool without decreasing the quality of the alignment. The mapping step could also take into account the read-pairing information to provide more precise alignments. From a more theoretical viewpoint, a study of whether using successively multiple k-mer sizes provides an even better correction (albeit at the price of a longer running time) would be an interesting perspective. Last but not least, another possible development could look into applications to datasets with highly heterogeneous coverage, as observed in single-cell, transcriptome or metagenome sequencing data.

4.2 Perspectives regarding long reads

Surprisingly, the idea of aligning reads on a DBG was applied to long, noisy reads before short, accurate ones. The efficiency of LoRDEC (Salmela and Rivals, 2014) and Bcool, respectively on long and short reads, suggests that such DBG-based correction is a general approach that can be applied to various kinds of datasets. Short reads can also be used in conjunction with long reads, as for correcting systematic errors such as ONT homopolymers (Jain *et al.*, 2016).

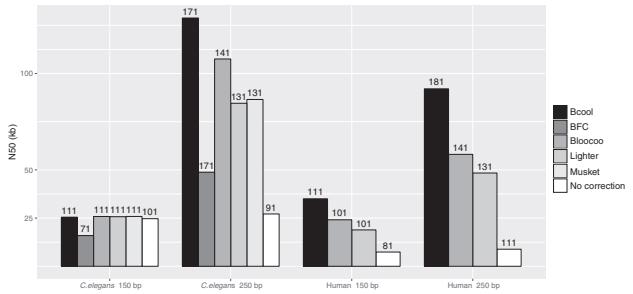

Fig. 6. Comparison of the N50s of the best assemblies obtained from the corrected simulated reads using Minia (Chikhi and Rizk, 2013). Only the best assembly obtained was kept and the corresponding *k*-mer size is indicated above each bar. BFC and Musket ran out of memory on all full human datasets

Table 2. Results of assembling the Illumina dataset DRR05374 (*C.elegans*) using fermi following various read correctors

Corrector	N50	NGA50	N75	NGA75	Nb contigs
Bcool	**23 021**	**20 696**	**10 880**	**8552**	8373
BFC	19 524	17 872	9490	7642	9235
Bloocoo	20 968	18 960	10 105	7993	8836
Lighter	21 435	19 269	10 254	8122	8714
Musket	22 545	20 291	10 788	8545	**8368**

Note: The best results among correctors are in bold.

Table 3. Results of assembling the Illumina dataset ERR2173372 (*A.thaliana*) using fermi following various read correctors

Corrector	N50	NGA50	N75	NGA75	Nb contigs
Bcool	**34 206**	**24 466**	**16 690**	**9301**	8419
BFC	29 002	21 634	15 363	9254	**8376**
Bloocoo	27 737	20 751	14 281	8783	8783
Lighter	25 274	19 325	13 194	8277	9420
Musket	27 348	20 453	14 021	8628	9139

Note: The best results among correctors are in bold.

Using nearly perfect short reads as those corrected by Bcool may improve long-read correction. To test this, we simulated both short and long reads from the *C.elegans* reference genome and compared the amount of errors still present in the long reads after LoRDEC hybrid correction by mapping them on the reference using BWA. A coverage of 100× of short reads of 150 bp were simulated along with long reads with a 12% error rate using PBSIM (Ono *et al.*, 2013). Applying LoRDEC using non-corrected short reads lead to a 3.04% error rate on corrected long reads. When the short reads were first corrected using Bcool, the error rate on the corrected long reads fell to 2.33% (a 30.5% improvement). Additional work will be required to explore this idea further.

Last but not least, in the current context of decreasing error rates for long reads, we may soon reach a point at which *k*-mer or DBG-based techniques will manage to perform efficient *de novo* reference-based correction using long reads alone. LORMA (Salmela *et al.*, 2017) is a first such attempt at using DBG created from long reads to perform pure correction in an iterative manner. This suggests that DBGs still have a bright future in bioinformatics.

Funding

We thank Romain Feron and Camille Marchet for their support and implication in this project. A.L.'s postdoctoral position was funded by the Fédération Wallonie-Bruxelles via an 'Action de Recherche Concertée' (ARC) grant to Jean-François Flot. Computational resources were provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) [2.5020.11], as well as by GenOuest platform (genouest.org). This work was partially supported by the French ANR-14-CE23-0001 Hydrogen Project.

Conflict of Interest: none declared.

References

Akogwu,I. *et al.* (2016) A comparative study of *k*-spectrum-based error correction methods for next-generation sequencing data analysis. *Hum. Genomics*, **10**, 20.

Benoit,G. *et al.* (2014) Bloocoo, a memory efficient read corrector. In: *European Conference on Computational Biology (ECCB)*. https://github.com/GATB/bloocoo.

Chikhi,R. *et al.* (2015) On the representation of de Bruijn graphs. *J. Comput. Biol.*, **22**, 336–352.

Chikhi,R. *et al.* (2016) Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, **32**, i201–208.

Chikhi,R. and Medvedev,P. (2014) Informed and automated *k*-mer size selection for genome assembly. *Bioinformatics*, **30**, 31–37.

Chikhi,R. and Rizk,G. (2013) Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms Mol. Biol.*, **8**, 22.

Dekker,J. *et al.* (2002) Capturing chromosome conformation. *Science*, **295**, 1306–1311.

DePristo,M.A. *et al.* (2011) A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat. Genet.*, **43**, 491–498.

Flot,J.-F. *et al.* (2015) Contact genomics: scaffolding and phasing (meta)genomes using chromosome 3D physical signatures. *FEBS Lett.*, **589**, 2966–2974.

Gurevich,A. *et al.* (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.

Jain,M. *et al.* (2016) The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biol.*, **17**, 239.

Kitzman,J.O. (2016) Haplotypes drop by drop: short-read sequencing provides haplotype information when long DNA fragments are barcoded in microfluidic droplets. *Nat. Biotechnol.*, **34**, 296–299.

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.

Li,H. (2012) Exploring single-sample SNP and INDEL calling with whole-genome *de novo* assembly. *Bioinformatics*, **28**, 1838–1844.

Li,H. (2015) BFC: correcting Illumina sequencing errors. *Bioinformatics*, **31**, 2885–2887.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Li,R. *et al.* (2009) SNP detection for massively parallel whole-genome resequencing. *Genome Res.*, **19**, 1124–1132.

Limasset,A. *et al.* (2016) Read mapping on de Bruijn graphs. *BMC Bioinform.*, **17**, 237.

Limasset,A. *et al.* (2017) Fast and scalable minimal perfect hashing for massive key sets. In: Iliopoulos,C.S. *et al.* (eds) *Proceedings of the 16th International Symposium on Experimental Algorithms (SEA 2017), London, UK, June 21-23, 2017, Leibniz International Proceedings in Informatics Volume 75*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany, pp. 25:1–25:16.

Liu,Y. *et al.* (2013) Musket: a multistage *k*-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics*, **29**, 308–315.

Loman,N.J. *et al.* (2015) A complete bacterial genome assembled *de novo* using only nanopore sequencing data. *Nat. Methods*, **12**, 733–735.

Marie-Nelly,H. *et al.* (2014) High-quality genome (re)assembly using chromosomal contact data. *Nat. Commun.*, **5**, 5695.

Mohamadi,H. *et al.* (2017) ntCard: a streaming algorithm for cardinality estimation in genomics data. *Bioinformatics*, **33**, 1324–1330.

Nagarajan,N. and Pop,M. (2009) Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *J. Comput. Biol.*, **16**, 897–908.

Ono,Y. *et al.* (2013) PBSIM: PacBio reads simulator—toward accurate genome assembly. *Bioinformatics*, **29**, 119–121.

Pevzner,P.A. *et al.* (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci.*, **98**, 9748–9753.

Salmela,L. and Rivals,E. (2014) LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, **30**, 3506–3514.

Salmela,L. and Schröder,J. (2011) Correcting errors in short reads by multiple alignments. *Bioinformatics*, **27**, 1455–1461.

Salmela,L. *et al.* (2017) Accurate self-correction of errors in long reads using de Bruijn graphs. *Bioinformatics*, **33**, 799–806.

Schröder,J. *et al.* (2009) SHREC: a short-read error correction method. *Bioinformatics*, **25**, 2157–2163.

Schwager,E.E. *et al.* (2017) The house spider genome reveals an ancient whole-genome duplication during arachnid evolution. *BMC Biol.*, **15**, 62.

Song,L. and Florea,L. (2015) Rcorrector: efficient and accurate error correction for Illumina RNA-seq reads. *GigaScience*, **4**, 48.

Song,L. *et al.* (2014) Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biol.*, **15**, 509.

Wick,R.R. *et al.* (2017) Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS Comput. Biol.*, **13**, e1005595.

Yang,X. *et al.* (2013) A survey of error-correction methods for next-generation sequencing. *Brief. Bioinform.*, **14**, 56–66.

Yeo,S. *et al.* (2017) ARCS: scaffolding genome drafts with linked reads. *Bioinformatics*. **34**, 725–731.