Inference of viral quasispecies with a paired de Bruijn graph

Freire, Borja

2021-02-15

Freire , B , Ladra , S , Parama , J R & Salmela , L 2021 , ' Inference of viral quasispecies with a paired de Bruijn graph ' , Bioinformatics , vol. 37 , no. 4 , pp. 473-481 . https://doi.org/10.1093/bioinformatics/b

http://hdl.handle.net/10138/341190
https://doi.org/10.1093/bioinformatics/btaa782

acceptedVersion

Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Original Paper

Inference of viral quasispecies with a paired de Bruijn graph

Borja Freire [1], Susana Ladra [1], Jose Paramá [1,*], and Leena Salmela [2]

[1] Universidade da Coruña, Centro de investigación CITIC, Facultade de Informática, A Coruña, Spain
[2] Department of Computer Science, Helsinki Institute for Information Technology, University of Helsinki, Helsinki, Finland.

[*] Corresponding author.

Associate Editor: Bonnie Berger

Received on June 13, 2019; revised on November 15, 2019; accepted on August 31, 2020

Abstract

Motivation: RNA viruses exhibit a high mutation rate and thus they exist in infected cells as a population of closely related strains called viral quasispecies. The viral quasispecies assembly problem asks to characterise the quasispecies present in a sample from high-throughput sequencing data. We study the *de novo* version of the problem, where reference sequences of the quasispecies are not available. Current methods for assembling viral quasispecies are either based on overlap graphs or on de Bruijn graphs. Overlap graph based methods tend to be accurate but slow, whereas de Bruijn graph based methods are fast but less accurate.
Results: We present viaDBG, which is a fast and accurate de Bruijn graph based tool for *de novo* assembly of viral quasispecies. We first iteratively correct sequencing errors in the reads, which allows us to use large k-mers in the de Bruijn graph. To incorporate the paired-end information in the graph, we also adapt the paired de Bruijn graph for viral quasispecies assembly. These features enable the use of long range information in contig construction without compromising the speed of de Bruijn graph based approaches. Our experimental results show that viaDBG is both accurate and fast, whereas previous methods are either fast or accurate but not both. In particular, viaDBG has comparable or better accuracy than SAVAGE, while being at least nine times faster. Furthermore, the speed of viaDBG is comparable to PEHaplo but viaDBG is able to retrieve also low abundance quasispecies, which are often missed by PEHaplo.
Availability: viaDBG is implemented in C++ and it is publicly available at `https://bitbucket.org/bfreirec1/viadbg`. All data sets used in this article are publicly available at `https://bitbucket.org/bfreirec1/data-viadbg/`.
Contact: jose.parama@udc.es
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

RNA viruses such as the human immunodeficiency virus (HIV), the Zika virus, and the hepatitis C virus (HCV) exhibit a high mutation rate (Duffy *et al.*, 2008). Thus their populations in a host organism consist of a number of different strains which are differentiated from each other by mutations in the genome. In the context of viruses, the collection of these strains is called a viral quasispecies (Domingo *et al.*, 2012; Holmes, 2009). Each of the strains in the viral quasispecies can be characterised by its haplotypic sequence. When studying a viral sample, it is important to capture all strains present in the sample, because different viral strains may have a different response to the available treatments and drugs (Domingo *et al.*, 2012).

High-throughput sequencing has provided a way to investigate viral samples in detail to characterise the different strains present in the sample and their abundances. However, although viral genomes are short, there are challenges that are specific to the analysis of viral quasispecies data. First, the presence of similar strains in the data makes it difficult to assign the reads to different haplotypic sequences. Secondly, viral samples are typically sequenced to a much deeper coverage than e.g samples for genomic or metagenomic sequencing. This presents a challenge for developing computationally efficient tools for reads that frequently overlap each other. Therefore on viral samples, standard tools for genome assembly or metagenomics produce fragmented assemblies that do not properly capture all strains present in the sample (see e.g. Baaijens *et al.* (2017)).

Methods for assembling viral quasispecies from high-throughput sequencing data are classified into two approaches, referenced based and *de novo* approaches (Posada-Cespedes *et al.*, 2017). The reference based approaches first align the reads to the reference sequence. Many of these approaches then cluster the reads to haplotypes by enumerating maximum cliques (Töpfer *et al.*, 2014), assembling the reads (Jayasundara *et al.*, 2015), using Hidden Markov Models (Töpfer *et al.*, 2013), or using probabilistic modelling (Prabhakaran *et al.*, 2014; Zagordi *et al.*, 2011; Ahn and Vikalo, 2018; Barik *et al.*, 2018). Instead of clustering reads, Knyazev *et al.* (2019) cluster the observed variants to haplotypes. Prosperi and Salemi (2011) divide the reference into overlapping intervals, construct local haplotypes for each interval, and finally merge them to global haplotypes. These reference-based approaches can be effective if a good quality reference is available. However, it has been shown that using reference genomes can bias the reconstruction significantly (Baaijens *et al.*, 2017; Töpfer *et al.*, 2014). Thus a number of *de novo* viral quasispecies assemblers, which do not need a reference sequence, have been developed. We are aware of three tools fitting this category, MLEHaplo (Malhotra *et al.*, 2015), SAVAGE (Baaijens *et al.*, 2017), and PEHaplo (Chen *et al.*, 2018). The *de novo* assemblers typically cannot assemble each strain into a single haplotype but instead produce a set of contigs. Baaijens *et al.* (2019) have recently proposed a method that takes as input contigs produced by a *de novo* viral quasispecies assembler and uses frequency information to further merge these into global haplotypes. In this work, we focus on the *de novo* contig assembly of viral quasispecies data.

Similar to the most successful genome assemblers for bacterial and eukaryotic genomes, *de novo* viral quasispecies assemblers use either an overlap graph or a de Bruijn graph to represent the sequencing data. See e.g. Nagarajan and Pop (2013) for a discussion on genome assembly approaches. An overlap graph is constructed by finding all pairwise overlaps between the sequencing reads. Given the deep sequencing of viral data, the number of actual overlaps between the reads approaches the quadratic worst case limit, and thus this step could be computationally expensive. On the other hand, methods based on overlap graphs such as SAVAGE produce very accurate assemblies, because the overlap graph captures well the long range similarities between the reads. PEHaplo introduces a different trade-off for overlap graph based approaches by introducing a technique to reduce the number of reads. It is thus much faster, but unfortunately also less accurate. The de Bruijn graph based methods such as MLEHaplo do not need to perform computationally intensive overlap computations between the reads. Instead they decompose the reads into k-mers and construct a de Bruijn graph where the $k-1$-mers are the nodes of the graph and an edge is added between two nodes if the corresponding k-mer is present in the read set. Because k-mers can be extracted by a linear scan over the reads, these approaches are computationally efficient. However, they are not able to optimally use long range information available in full length reads and thus the assemblies they produce tend to be more fragmented and less accurate.

High-throughput sequencing reads such as Illumina reads are typically paired-end reads. Many of the viral quasispecies assemblers use heuristics to incorporate the paired-end information. SAVAGE merges read pairs when the pairs overlap each other and it accepts overlaps involving paired-end reads only if both pairs are involved in the overlap and their orientation in the overlap is the same. PEHaplo uses heuristics to prune the overlap graph based on paired-end information and it uses paired-end information as a guidance for finding paths in the overlap graph. PEHaplo also includes a post assembly step where contigs are split based on paired-end alignments. MLEHaplo formulates the viral quasispecies assembly problem as finding a path cover with maximum score from paired-end reads in a de Bruijn graph. This problem is shown to be NP-hard and thus MLEHaplo implements a heuristic path finding algorithm for this problem.

We present viaDBG (viral assembly with paired de Bruijn Graph), a fast and accurate tool for viral quasispecies assembly. Our method is based on de Bruijn graphs (DBGs), which we augment with several techniques to improve the accuracy of the reconstructed haplotypic sequences. First we employ an iterative error correction method with increasing k-mer sizes. This allows us to use large k-mers in the final assembly enabling the use of long range information in the de Bruijn graph. Furthermore, we adapt the approximate paired de Bruijn graph (APDB) (Medvedev *et al.*, 2011) to viral quasispecies assembly. Whereas almost every assembler nowadays applies paired-end information in the post-processing phase, where contigs are merged and/or topologically sorted to create scaffolds, in the APDB the paired-end information is added to the DBG.

Our experiments show that on both synthetic and real data viaDBG is among both the most accurate methods and the fastest methods, whereas previous tools are either accurate or fast but not both. For example, viaDBG is up to 43 times faster than SAVAGE and produces assemblies with comparable accuracy. Furthermore, viaDBG is able to recover also low abundance strains which are lost or inaccurately assembled by PEHaplo, while matching the speed of PEHaplo. On real sequencing data, viaDBG produces assemblies with three times as high N50 values as SAVAGE while being nine times faster. The speed of viaDBG is comparable to PEHaplo on this data set but depending on whether we look at polished or unpolished contigs, PEHaplo either mixes the strains or produces a 30% lower N50 value than viaDBG, while viaDBG produces accurate results. The de Bruijn graph based approach makes viaDBG efficient, whereas the accuracy of viaDBG is due to using a large k in the de Bruijn graph and the systematic use of paired-end information.

2 Methods

2.1 Background

2.1.1 Error correction by LoRDEC

LoRDEC (Salmela and Rivals, 2014) is a hybrid error correction method for correcting sequencing errors in *Third Generation Sequencing* reads with the help of accurate short reads. LoRDEC defines a k-mer as solid if it occurs at least t times in the short read data where t is the abundance threshold. The solid k-mers are then used to build a DBG. The third generation sequencing reads are then processed one at a time. First, solid k-mers in the read are identified and the regions between solid k-mers are called weak. Then for each weak region between two solid k-mers, LoRDEC finds the best matching path in the DBG between the two solid k-mers. This path is used to correct the weak region in the read. Finally, the weak ends of the read are aligned to the DBG starting from the extremal solid k-mer and the weak ends are corrected according to the found paths. To limit the runtime of the method, LoRDEC abandons the search for the best alignment if there are too many branches in the DBG.

2.1.2 Approximate paired de Bruijn graph

Medvedev *et al.* (2011) presented the approximate paired de Bruijn graph (APDB) to leverage paired-end information directly in contig assembly. To build the APDB, they first extract all bilabels from the paired-end reads. A bilabel is a pair of k-mers (A, B) such that A occurs in position p in a left-hand read and B occurs in position p in the corresponding right-hand read. Two bilabels (A, B) and (C, D) are merged if $A = C$ and B is reachable from D or vice versa. The merged bilabels form the edges of the APDB and thus the edges have the form (A, S) where A is a k-mer and S is a set of k-mers. An edge (A, S) connects two nodes, $(\mathrm{pref}(A), \mathrm{pref}(S))$ and $(\mathrm{suf}(A), \mathrm{suf}(S))$, where $\mathrm{pref}(A)$ $(\mathrm{suf}(A))$ is the $k-1$ length prefix (suffix) of the k-mer A and $\mathrm{pref}(S)$ $(\mathrm{suf}(S))$ is the set of $k-1$ length prefixes (suffixes) of all the k-mers in the set S.

Unfortunately, APDB is not directly applicable to viral quasispecies assembly. Consider a case where the left hand reads derive from a region of the genome where two strains are equal and the right hand reads derive from a region where the strains differ by a single SNP. When we extract

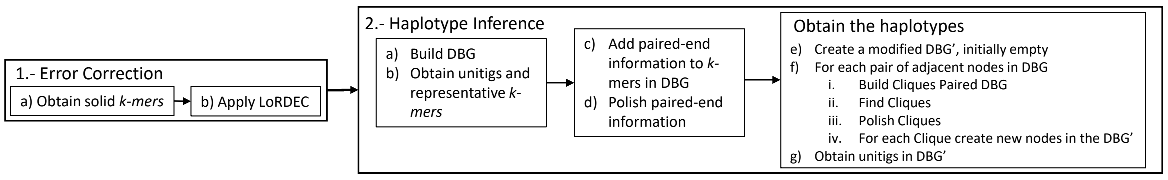

1.- Error Correction
a) Obtain solid *k-mers*
b) Apply LoRDEC
2.- Haplotype Inference
a) Build DBG
b) Obtain unitigs and representative *k-mers*
c) Add paired-end information to *k*-mers in DBG
d) Polish paired-end information
Obtain the haplotypes
e) Create a modified DBG', initially empty
f) For each pair of adjacent nodes in DBG
i. Build Cliques Paired DBG
ii. Find Cliques
iii. Polish Cliques
iv. For each Clique create new nodes in the DBG'
g) Obtain unitigs in DBG'

Fig. 1. Overview of viaDBG. Our method has two main steps, error correction and haplotype inference. The error correction step aims to correct the sequencing errors in the reads by first identifying solid k-mers in the reads and then applying the LoRDEC algorithm. The haplotype inference step starts by building a DBG and obtaining unitigs. The paired-end information is then added to the DBG and some heuristics are used to polish the paired-end information. Finally, the haplotypes are obtained by splitting the DBG nodes based on the paired-end information and obtaining unitigs from this modified DBG.

bilabels from these reads, the left k-mers will be the same in both strains but the extracted right k-mers can be the same or different depending on whether they cover the SNP or not. Let us suppose that we have extracted bilabels (A, B), (A, B'), and (A, C) from the reads where B and C occur in the first strain and B' and C in the second strain. Because B is reachable from C and B' is reachable from C, all these bilabels are merged into a single edge in APDB. This is acceptable if the goal is to construct a single genomic sequence but not for viral quasispecies assembly where we need to construct all the strains. Here we have devised a method to differentiate such bilabels correctly. The key idea is to merge a set of bilabels only if all right-hand k-mers are pairwise reachable from each other.

2.2 Overview of our method

Figure 1 shows the main steps followed by viaDBG. The main difference with respect to typical assembly methods based on de Bruijn graphs is the use of paired-end information in an early stage. Paired-end reads are composed by two reads, which are the two extremes (left- and right-hand) of a sequencing fragment. The insert size is the number of base pairs between the two reads. We will use Δ to denote the maximum error in the insert size.

2.3 Error correction

The error rate of paired-end short reads is low, which makes them suitable for assembly methods based on the DBG. It has been shown that longer k-mers lead to better assembly, but the probability of getting erroneous k-mers also increases. Therefore, we devote the first step to remove sequencing errors from reads, to obtain longer correct k-mers, and thus reducing the erroneous information that would lead to shorter contigs, lower genome fraction recovered and/or more misassemblies.

The error correction involves two steps: I) selection and classification of solid k-mers that, as in the case of LoRDEC, are the k-mers whose abundance in the reads is higher than a threshold, and II) reads correction.

2.3.1 Selection of solid k-mers

A k-mer is genomic if it appears in at least one strain and a non-genomic k-mer does not appear in any of the strains in the sample. As in LoRDEC we select solid k-mers based on their frequency of appearance, assuming that genomic k-mers are more frequent than the non-genomic ones. Then, the whole read set is traversed and each k-mer is classified as solid or not solid. Because of the conservative selection of the threshold, we expect the non-solid regions to be compounded with erroneous information. This is simple, but the problem is to determine the threshold.

In our work, the search of that value is based on the following idea. Let us first consider the histogram of the number of different k-mers that occur at each frequency, that is, for each frequency f_i, we plot $n(f_i)$, which is the number of different k-mers occurring f_i times. Then, we expect to find a change in the trend in the histogram among the number of different k-mers having low frequencies (non-genomic k-mers) and those having higher frequencies (genomic k-mers). On one hand, the number of different non-genomic k-mers decreases as the frequency increases. On the other hand, the number of different genomic k-mers, which have higher frequencies, starts outnumbering the number of different non-genomic k-mers. Thus, we will use the starting position of that change of trend in the histogram as the threshold to detect solid k-mers.

More concretely, we will search for a region in the histogram where there is a frequency whose count is lower than most of the counters for the frequencies in the succeeding zone. We make use of a fixed window size N to find the frequency f_t where that change of trend starts. We define t as the smallest i such that $f_i \geq 1$ and

$$|\{f_j | f_i \leq f_j \leq f_{i+N} \text{ and } n(f_j) > n(f_i)\}| \geq N/2.$$

In the Supplementary Material we show that the choice of N is easy, by showing that with a wide range of different window values, the performance of viaDBG does not differ significantly. By default we use $N = 16$.

2.3.2 Error correction algorithm

To correct sequencing errors in the read, we adapted the LoRDEC algorithm (Salmela and Rivals, 2014) for viral quasispecies data. We use the solid k-mers identified above to build a DBG and then align all the reads to the DBG to correct them. If the abundance of a strain is such that the corresponding k-mers are solid, the strain is present as a path in the DBG. The reads are corrected by aligning them to this graph and choosing the alignment with the smallest edit distance between the read and path in the graph. The reads are expected to align best against the path representing the strain they derive from and thus most of them are corrected towards the correct haplotypic sequence. Therefore this algorithm is well suited for correction of viral quasispecies data.

We made three further changes to better adapt the algorithm for viral quasispecies data. First, after building the DBG using the solid k-mers, we polish it by removing short tips, i.e. short paths where the first node has outdegree larger than one and the last node has outdegree zero. Secondly, we only correct the part of the reads between the leftmost and rightmost solid k-mer because the algorithm is less accurate on the read ends when only one end of the alignment is anchored on solid k-mers. Third, in viral quasispecies data it is not necessary to abandon the search for best alignment if there are too many branches in the DBG because the genomes are smaller and the DBG is less tangled. Therefore this limitation was removed from the algorithm.

Once reads have been corrected, the k-mer size is doubled and the reads are corrected again. This process is repeated three times. In the first iteration when k is small, the set of solid k-mers contains most genomic k-mers and some non-genomic k-mers that are caused by the same sequencing error occurring in the same locus in several reads. Still our method can correct most errors already at this stage because most k-mers including an error are unique even for a small k. Once most errors have been corrected in the first iteration, we can increase k because longer k-mers are now expected to be correct. Because longer k-mers span more

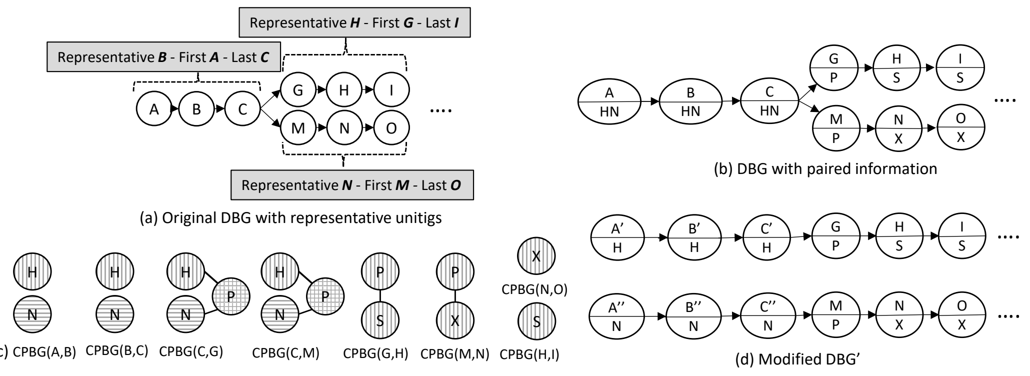

Representative *H* - First *G* - Last *I*
Representative *B* - First *A* - Last *C*
Representative *N* - First *M* - Last *O*
(a) Original DBG with representative unitigs
(b) DBG with paired information
(c) CPBG(A,B) CPBG(B,C) CPBG(C,G) CPBG(C,M) CPBG(G,H) CPBG(M,N) CPBG(H,I) CPBG(N,O)
(d) Modified DBG'

Fig. 2. Example of the different steps of haplotype inference. (a) First we build a DBG using all solid k-mers in the reads. Unitigs are then identified and a representative k-mer is assigned to each unitig. (b) Next we augment the graph with the paired-end information. P, S, and X are representative k-mers of unitigs not shown in the figure. (c) A Cliques Paired DBG (CPBG) is built for each adjacent pair of k-mers in the DBG. Nodes of CPBG are the paired k-mers of the adjacent pair of k-mers and edges between the nodes are added if there is a path between the corresponding k-mers in the DBG. (d) Finally for each CPBG we find cliques and each clique is used to split the nodes of the DBG.

variants, k-mers originating from different strains are separated better from each other. Thus also the abundance of erroneous k-mers becomes lower and now less of the erroneous k-mers are classified as solid. This allows us to further correct some sequencing errors in the next iterations.

2.4 Haplotype inference using paired-end reads

As seen in Figure 1, the haplotype inference is applied in several steps. To illustrate them, Figure 2 shows an example workflow.

First, a regular DBG is built with the solid k-mers obtained in the previous step. Then, we retrieve the unitigs of the DBG, and for each unitig, we assign a representative k-mer, which will be used later in the process. Next, each k-mer is associated with a set of paired k-mers. Given a k-mer A and a paired-end read where A appears at position p of the left-hand read, B is a paired k-mer of A if B occurs at position p of the right-hand read. Next, we polish the paired-end information, and finally we modify the DBG. If all occurrences of the k-mer A are from the same strain, then all paired k-mers of A occur along some path in the DBG. Thus they are all reachable from each other. On the other hand, if the k-mer A occurs in several strains, then the paired k-mers are likely to span some site containing a mutation. Note that the paired k-mers span an area larger than k in the haplotypic sequences. Therefore, they are not all reachable from each other but it still holds that the paired k-mers originating from the same strain are all reachable from each other. We will use this reachability information to split the DBG nodes into different strains. Finally, the contigs are retrieved from this new DBG. Next we explain each of these steps in detail.

2.4.1 Getting unitigs and representative k-mers

A unitig is a unary path in the de Bruijn graph, that is, a path where all nodes have in-degree and out-degree equal to one except for the first and last nodes. Unitigs always belong to the final genome/s. Therefore, some assemblers, such as SPAdes (Bankevich *et al.*, 2012), condense unitigs into single nodes to compact the graph without losing information.

In our case, for each unitig, we extract three elements: first, middle and last k-mers. The middle k-mer serves as a *representative* of the unitig, whereas the first and last k-mers are used to determine if there is a path from one unitig to another.

Figure 2(a) shows the DBG of our running example where unitigs are delimited by a brace. The representative k-mer and the first and last k-mers are also displayed in a grey box.

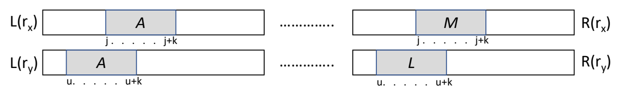


Fig. 3. Extracting paired k-mers from paired-end reads. *P(A)=(M,L).*

2.4.2 Adding paired-end information for each k-mer

One of the key features of our method is the use of the paired-end information, as it provides additional clues of the actual strains during the traversal of the DBG.

Let U be the set of paired-end reads. Given a read $r \in U$, $L(r)$ is the left-hand, $R(r)$ is the right-hand read, and $L(r)[l \dots m]$ ($R(r)[l \dots m]$) are the base pairs at positions $l \dots m$ of $L(r)$ ($R(r)$). For each k-mer A, our method needs to compute its set of paired k-mers $P(A) = \{M \mid M$ is a solid k-mer and, $\exists\, r_x \in U$ and a position j such that $L(r_x)[j \dots j+k-1] = A$ and $R(r_x)[j \dots j+k-1] = M\}$.

Figure 3 shows an example where the k-mer A appears in the left-hand part in two reads (r_x and r_y). Then the solid k-mers M and L, which appear in the same positions of the right-hand parts, form $P(A)$.

To avoid excessive memory usage, we do not store all paired k-mers. Instead, for each k-mer in $P(A)$, we find the unitig to which it belongs and replace that k-mer with the representative k-mer of the unitig.

Observe in Figure 2(b), for example, that k-mer A has two paired k-mers H and N, which are the representative k-mers of the unitigs GHI and MNO, respectively.

2.4.3 Polishing paired-end information

In this step, for each solid k-mer A, its $P(A)$ is polished. This is needed since sometimes the variance of the insert size can be larger than the used Δ, as the insert size distribution can be modelled with a normal distribution. Therefore, we design a paired-end polishing method that removes outliers with large variance in the insert size, while avoiding the removal of low abundance strains.

Let $\text{freq}(A, M)$ be the frequency of the appearance of the k-mer pair (A, M) in paired-end reads $r \in U$ such that $M \in P(A)$. Because the insert size is normally distributed, the frequency of a k-mer pair with insert size close to the mean is expected to have a high frequency, whereas a k-mer pair with insert size far from the mean is expected to have a low frequency. Furthermore, if the insert size of a k-mer pair (A, M) is close to the mean, then within a short distance from the node corresponding to M in the DBG, we expect to see many other k-mers L that are also in $P(A)$ and have a frequency $\text{freq}(A, L) \geq 1$. Again, this is not expected for a k-mer pair whose insert size is far from the mean. We combine these two effects into a smoothed frequency $\text{freq}'(A, M)$, which is defined as follows:

$$\text{freq}'(A,M) = \min \begin{cases} \text{freq}(A,M) + & |\{L \mid \text{freq}(A,L) \geq 1 \\ & \quad \text{and } d(M,L) < \text{max-path-len}\}| \\ \text{max-threshold} \end{cases}$$

where $d(M, L)$ denotes the distance between M and L in the DBG and we set max-threshold to 40 and max-path-len to 20. Finally, we keep only those paired k-mers whose frequency is within top 85%. We experimentally found that those values work well in practice for all cases.

We limit the smoothed frequency by max-threshold to preserve low abundance haplotypes. Without such limit, the frequency of paired k-mers of high abundance strains with higher divergence from the mean insert size often gets higher than the frequency of paired k-mers of low abundance strains with insert size close to the mean value. This is especially important when relative abundances are around 1–2%.

Currently, this step is the bottleneck of the algorithm. We need to compute the distance between $\frac{n(n-1)}{2}$ pairs per node, where n is the number of pairs in the list of paired k-mers of a given k-mer.

2.4.4 Obtaining the haplotypes

This subsection describes in detail the third block of Step 2 of Figure 1 (steps labelled 2.(e), 2.(f), and 2.(g)). The haplotypes are obtained by splitting the nodes of the DBG built in Step 2.(a), based on the paired-end information and obtaining unitigs from this modified DBG. Therefore, the Step 2.(e) starts by creating a new empty DBG'.

- *Step 2.(f) i*: For each pair of adjacent nodes *(A,B)* of the DBG, a *Cliques Paired de Bruijn Graph* (CPBG) graph is built. *CPBG(A,B)* is an undirected graph. The paired k-mers of A and B are the nodes of *CPBG(A,B)*, i.e. the set of nodes is $P(A) \cup P(B)$. There is an edge between two nodes U, V in the *CPBG(A,B)* if there is a path of length $\leq 2\Delta$ in the DBG from $last(U)$ to $first(V)$ or from $last(V)$ to $first(U)$.[1]

  Observe that we are computing the CPBG of k-mers A and B that are adjacent in the DBG. Therefore their paired k-mers (separated from A and B, on average, by the insert size) would also be neighbours in the DBG since they ideally differ by 1 base pair as well, or they would be very close to each other, due to the insert size error, forward and backward, that is, 2Δ. Therefore, in *CPBG(A,B)*, we link the paired k-mers of A and B that are connected by a path of the DBG of size at most 2Δ.

  Figure 2(c) shows the CPBG of all pairs of adjacent nodes in the DBG of our example. For example, observe in *CPBG(A,B)* that the nodes are the paired k-mers of A and B, which are H and N in both cases. However, there is no path in the DBG connecting H and N, and thus, in the CPBG there is not an edge linking them. In the case of *CPBG(C,G)*, there is an edge between H and P, since there is a path of length at most 2Δ in the DBG that connects them (not shown in the DBG of Figure 2 to avoid cluttering the figure). Similarly, there is an edge connecting N and P.

  Here we can see the other important benefit of using representatives. Observe that in order to determine whether there is an edge connecting a pair of nodes U and V of a CPBG, the algorithm has to find paths in the DBG, between the unitigs of the DBG corresponding to U and V. Therefore, decreasing the number of nodes of the CPBG speeds up this process.
- *Step 2.(f) ii*: For each CPBG, we obtain all its maximal cliques. A clique is a set of nodes of the graph where all nodes are connected to each other.

  In Figure 2(c), observe the *CPBG(C,G)*. There are two cliques; the first one is formed by H and P, and the other by N and P.

  Conceptually, cliques are sets of k-mers that belong to the same haplotypic sequence. Since all the paired-end k-mers in the clique reach or are reached by others in the DBG, it means that there is one strain that gathers them together.

  However, when the graph is tangled, it is possible to find paths in the DBG for two k-mers that do not belong to the same strain, and this may produce fake cliques. Therefore, we select the cliques that are supported by the frequency of appearance of their k-mers, more precisely, we select those cliques whose nodes appear in more reads and are more linked to other nodes in the DBG. Full details of this process are given in Section 2 in the Supplementary Material. Choosing a large value of k makes the graph less tangled and thus alleviates this problem. Also using a small Δ helps because even in a tangled graph, shorter paths are less likely to be incorrect.

  We obtain another benefit by using representative k-mers, since the CPBG is not a large graph, maximal cliques can be found with lower computational cost than in the case of using all k-mers.
- *Step 2.(f) iii*: Because of errors in reads, repetitive sections and shared strain regions, wrong cliques can be created. We use several heuristics to polish the cliques.

  – We remove small cliques because they often rise from erroneous k-mers.
  – Shared strain regions produce cliques where all k-mers are paired with A while a subset of them is also paired with B, that is, all nodes of the clique are in $P(A)$, and some nodes, but not all, are in $P(B)$. To keep strains with shared regions separate, we also remove these cliques.
  – Let $\mathcal{SC}$ be the set of all cliques found so far. When two cliques $\mathcal{C}\ell_x,\ \mathcal{C}\ell_y \in \mathcal{SC}$ are almost the same, we remove the smallest one because such cliques can arise from sequencing errors. More precisely two cliques are considered almost the same when $|(\mathcal{C}\ell_x \cap \mathcal{C}\ell_y)| \geq R * \min(|\mathcal{C}\ell_x|, |\mathcal{C}\ell_y|)$, where R is a threshold value. By default we use $R = 90\%$.

- *Step 2.(f) iv*: For each pair of adjacent nodes A and B in DBG, we take the set of cliques $\mathcal{SC}_{CPBG(A,B)}$ of *CPBG(A,B)* and, for each clique $\mathcal{C}\ell_x \in \mathcal{SC}_{CPBG(A,B)}$: If $\mathcal{C}\ell_x$ has nodes of $P(A)$ and $P(B)$, then the nodes $A_{P_A \cap \mathcal{C}\ell_x}$ and $B_{P_B \cap \mathcal{C}\ell_x}$ are added to *DBG'*, unless they are already in *DBG'*. $A_{P_A \cap \mathcal{C}\ell_x}$ is a node corresponding to the k-mer A having paired-end information $P_A \cap \mathcal{C}\ell_x$, similarly $B_{P_B \cap \mathcal{C}\ell_x}$ is a node corresponding to B having paired-end information $P_B \cap \mathcal{C}\ell_x$.

  In the case of nodes C and G of the example of Figure 2, their *CPBG(C,G)* has two cliques $\mathcal{C}\ell_1 = \{H, P\}$ and $\mathcal{C}\ell_2 = \{N, P\}$. Then, a new node C' is created due to the existence of $\mathcal{C}\ell_1$, with paired information $P(C) \cap \mathcal{C}\ell_1 = \{H, N\} \cap \{H, P\} = \{H\}$, as seen in Figure 2(d). C'' is derived from the clique $\mathcal{C}\ell_2$, thus this new node has as paired information $P(C) \cap \mathcal{C}\ell_2 = \{H, N\} \cap \{N, P\} = \{N\}$. Next G is processed accordingly, producing only one version with paired info P. These nodes are added to *DBG'*.

  Observe that, in *DBG'*, C' and C'' correspond to the same k-mer but those nodes have different paired information, which means that they correspond to different strains.
- *Step 2.(g)*: The last step of the algorithm enumerates the unitigs in the new *DBG'*. As a result of the adaptations based on the CPBG analysis, unitigs are expected to be much longer than in the previous DBG.

[1] $first(K)$ is the first k-mer of the unitig of which K is the representative, while $last(K)$ is the last k-mer.

Table 1. Main characteristics for the data sets with ground truth available.

	Virus Type	Genome Length (bp)	Average Coverage	Num. Strains	Abun-dance	Diver-gence
HIV-real	HIV-1	9487–9719	20000x	5	10–30%	1–6%
HIV-5	HIV-1	9487–9719	20000x	5	5–28%	1–6%
ZIKV-3	ZIKV	10251–10269	20000x	3	16–60%	3–10%
ZIKV-15	ZIKV	10251–10269	20000x	15	1–13%	1–12%
HCV-10	HCV-1a	9273–9311	20000x	10	5–19%	6–9%

3 Results

We compare viaDBG with previous methods for *de novo* viral quasispecies assembly. We also include SPAdes (Bankevich *et al.*, 2012) and metaSPAdes (Nurk *et al.*, 2017) in the comparison to show that viaDBG improves upon general approaches for genome assembly and metagenomic assembly in the case of viral data. We omit some comparisons, such as the reference-based approaches PredictHaplo (Prabhakaran *et al.*, 2014) and ShoRAH (Zagordi *et al.*, 2011), as Baaijens *et al.* (2017) have shown that SAVAGE outperforms both of them. We perform experiments both on simulated and real Illumina MiSeq data.

In the case of *de novo* viral quasispecies assemblers, we compared viaDBG with SAVAGE (Baaijens *et al.*, 2017), which has proven to be the most precise tool among the whole *de novo* assemblers for viral quasispecies, and with PEHaplo (Chen *et al.*, 2018), which is the last released state of the art tool. Real data was trimmed using CutAdapt (Martin, 2011), removing primers, low quality and extremely short reads. In the case of SAVAGE, whose authors highly encourage the usage of PEAR (Zhang *et al.*, 2014), it was only applied to the data set HIV-real described in Section 3.1.1 and to the data set HCV-10 described in Section 3.1.2, since for the rest of the data sets, SAVAGE could not complete the assembly - due to memory crash - when running on the result of applying PEAR. In the case of PEHaplo, PEAR was not applied since their authors discourage its usage. PEAR was not applied on synthetic data sets for viaDBG. However, we used PEAR on the real data sets (HIV-real and the real ZIKV and HCV samples) for viaDBG because the reads were shorter in these data sets and using PEAR ensured that we could use a large k for constructing the DBG.

3.1 Benchmarking data

In our experimental evaluation, we used both simulated and real MiSeq sequenced data. We have followed the methodology and data sets used by Baaijens *et al.* (2017), which are described next.

3.1.1 Real data with ground truth

We used a gold standard benchmark for viral assembly (Giallonardo *et al.*, 2014). The reads were produced from 5 HIV strains using Illumina MiSeq (2x250 bp with error around 0.3% and mean insert size 371 bp) with 20000x coverage. As the five strains contained in the sample are known, it is possible to validate the achieved results. Table 1 includes the main characteristics of this data set (HIV-real).

3.1.2 Synthetic benchmarks

Five different simulated data sets were used, consisting of 2x250 bp Illumina reads from different virus strains, namely human immunodeficiency virus (HIV), hepatitis C virus (HCV) and Zika virus (ZIKV). The HIV-5, ZIKV-3, and HCV-10 data sets are the data sets generated by Baaijens *et al.* (2017). The read length in these data sets is 2x250 bp and the insert size is 450 bp. The ZIKV-15 data set was regenerated by us using SimSeq with default configuration for Illumina MiSeq reads (read length 2x250 bp and insert size 500 bp). Table 1 also shows the main characteristics of these data sets.

3.1.3 Divergence ratio and relative abundance benchmarks

We used synthetic data sets for measuring the algorithm bounds. To analyse when the algorithm loses its effectiveness, we used data sets with extreme properties that differ from the real data or the synthetic data sets used in usual experiments, which are generally simulated using realistic properties. Thus, we used 36 data sets from HIV-86.9 strain, varying the divergence ratio (0.5%, 0.75%, 1%, 2.5%, 5%, and 10%) and the relative abundance (1:1, 1:2, 1:5, 1:10, 1:50, and 1:100) of each haplotype. These data sets also correspond to the data sets used by Baaijens *et al.* (2017), in an effort to avoid any bias in the experiments.

3.1.4 Real data without ground truth

We have included two real patient samples. More concretely, i) *Zika virus sample*: an Asian-lineage ZIKV sample consisting of Illumina MiSeq 2x300 bp reads with approximately 30000x coverage sequenced from a rhesus macaque after 4 days of infection (Dudley *et al.*, 2016) and publicly available in NCBI under the accession code SRR3332513, and ii) *Hepatitis C virus sample*: an HCV sample consisting of Illumina MiSeq reads with approximately 80000x coverage, sequenced from an Australian human patient after 135 days of infection, publicly available in NCBI under the accession code SRR1056035.

3.2 Evaluation scenarios

We ran several experiments under different scenarios. First, we analysed the behaviour of our method when the target genome is known, such that we can evaluate the obtained results. More concretely, we used the evaluator MetaQUAST (Mikheenko *et al.*, 2016) with the option "-unique-mapping", which allows us to assay metagenomic results getting the best unique alignment for each contig to the objective genomes, avoiding one contig to cover more than one genome fragment. We obtained several statistics, such as the largest contig, mismatches/indels/N-Rate, misassemblies, N50, genome fraction, etc. Furthermore, we measured the time spent and the memory used during the whole assembly process. As in previous work (Baaijens *et al.*, 2017), we only considered contigs above 500 bp.

To further analyse the performance of our method, we ran some experiments to check the algorithm bounds. We used the synthetic data with different abundance and divergence ratios, and compared the obtained results in terms of percentage of genome retrieved and percentage of mismatches.

Finally, we ran some experiments over those data sets with no available ground truth. This is the case for real virus sample HCV and ZIKV, which may have mixed data of other organisms different from the considered virus, making the discovery even more challenging.

3.3 Results comparison - overall performance

Table 2 shows a summary of the results obtained when applying each assembler over the benchmarking data sets. The complete table, including also the results for the data sets ZIKV-3 and HCV-10, and the additional values of number of contigs larger than 500 bp, length of the largest contig, percentage of indels, N-rate and total user CPU time, can be seen in the Supplementary Material.

Overall, the results show, as expected, that tools specifically designed for viral quasispecies inference obtain the best results in genome fraction and largest alignment for all data sets. SAVAGE, PEHaplo, and viaDBG show a good performance on the average length of the retrieved contigs. SAVAGE generally retrieves a higher genome fraction and obtains larger contigs than viaDBG and PEHaplo. When the data sets are more complex, namely large differences in genome abundances or high number of strains, PEHaplo fails. For example, we could not get meaningful results for PEHaplo on the ZIKV-15 data set and thus these are missing in Table 2. After correcting the reads, PEHaplo removes all of those that do not have

Table 2. Assembly results per method on the benchmarking data sets when ground truth is known. viaDGB* omits the correction step and PEHaplo** omits the polishing step.

data set	method	% genome	N50	misass-emblies	% mis-matches	elap time (min)	memory (GB)
HIV-real	viaDBG*	87.25%	1813	0	0.197	4.48	3.74
	viaDBG	89.53%	1986	0	0.204	20.01	3.74
	SAVAGE	91.79%	611	0	0.684	218.30	49.12
	PEHaplo	87.96%	2995	0	3.521	12.74	3.48
	PEHaplo**	91.43%	1262	0	0.074	7.56	3.48
	SPAdes	20.15%	660	1	2.091	12.74	5.52
	metaSPAdes	83.10%	1432	3	9.291	9.06	4.29
HIV-5	viaDBG	97.50%	8046	2	0.151	5.01	2.89
	SAVAGE	98.22%	6001	3	0.014	204.40	26.11
	PEHaplo	78.59%	9328	2	0.690	23.93	4.86
	SPAdes	90.91%	5097	2	0.051	3.31	4.12
	metaSPAdes	35.87%	6385	6	5.322	3.86	2.99
ZIKV-15	viaDBG	86.06%	1759	0	0.002	18.26	3.71
	SAVAGE	82.72%	1632	0	0.002	352.98	9.03
	PEHaplo	-	-	-	-	-	-
	SPAdes	38.97%	2063	0	0.147	6.17	4.42
	metaSPAdes	16.03%	3863	0	2.273	4.49	3.19

a large enough number of duplications or substrings. Ideally, when the data set has high coverage (around 20000x), every position of the genome will have a significant number of reads starting on it. However, when the number of strains is high, the coverage for each strain is reduced. Furthermore, if abundance for each strain is large, then the impact over the coverage for each strain is even higher. On the other hand, in accordance with the results reported by Baaijens *et al.* (2017), SPAdes gets results comparable with tools specifically designed for viral assembly on some of the simulated data sets, such as HIV-5, but poor performance over the real data set (HIV-real). Exactly the opposite happens with metaSPAdes, which obtains low genome fractions, large number of mismatches and low N50 values for simulated data, whereas it improves the genome fraction retrieved for HIV-real (while keeping high rates of mismatches and misassemblies).

Table 2 also shows that SPAdes' performance decreases when the number of strains increases, the relative abundance decreases, and similarity ratio increases. This is due to the fact that SPAdes does not implement any strategy to deal with this situation. As commented before, PEHaplo also encounters problems when the number of strains increases. In contrast, viaDBG and SAVAGE obtain similar performance, SAVAGE being more sensitive to the strain relative abundance.

On the HIV-real and HIV-5 data sets PEHaplo achieves the highest N50 but the contigs reported have much more errors than viaDBG (four times more mismatches on HIV-5 and 17 times more mismatches on HIV-real than the contigs produced by viaDBG). This indicates that some of the strains have been mixed in the contigs produced by PEHaplo. For the HIV-real data set we also report the results of PEHaplo without the polishing step (PEHaplo** in the table) and see that the mismatch rate is much lower without the polishing step but also the N50 drops below the N50 of viaDBG.

Focusing in the case of the real data set (HIV-real in Table 2), viaDBG has the overall best performance. Although the genome fraction retrieved is slightly lower than SAVAGE (2.26 percentage points), viaDBG is able to get the largest contig, a longer average contig, and a lower number of mismatches. Moreover, using the standard pipeline of PEHaplo, viaDBG obtains a larger genome fraction retrieved and a lower number of mismatches and indels. The high number of mismatches and indels obtained by the standard PEHaplo pipeline indicates that some of the haplotypes have been mixed. With this data set, PEHaplo obtains higher genome fraction and lower number of mismatches and indels, but also a lower N50 and shorter largest contig, if the polishing step is omitted (indicated as PEHaplo** in the table).

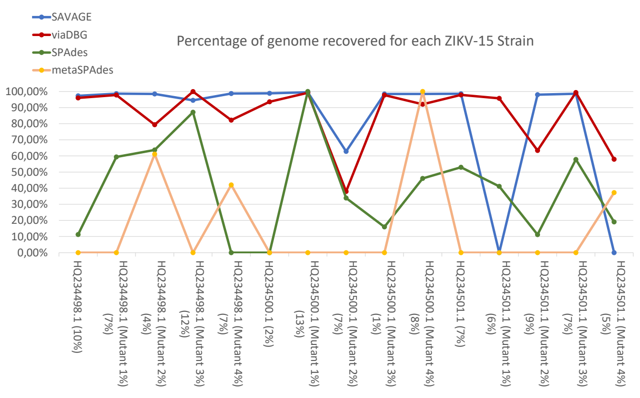


Fig. 4. Comparison between the four tools and the ZIKV-15 strains data set.

We will closely compare the behaviour of each method by taking into account the results for ZIKV-15 data sets, which can be considered the most challenging simulated data set. We omit PEHaplo from the comparison, as we were not able to run this tool and produce reliable results for this data set. Figure 4 shows the percentage of genome recovered for each strain of the ZIKV-15 data set. Table 2 shows that viaDBG and SAVAGE have a similar overall performance. However, a deeper comparison reveals that the behaviour of each method is rather different. SAVAGE retrieves the highest percentage of genome for most of the strains. This can be caused by the fact that viaDBG systematically removes the beginning and the end of the genomes due to lack of coverage in these regions. Despite SAVAGE outperforming viaDBG in most cases, SAVAGE fails at assembling the genome for two strains. This is not happening with viaDBG, which retrieves a significant portion of the genome in all cases, being close to SAVAGE in most cases, and even sometimes outperforming its results. More concretely, one of the genomes lost by SAVAGE, HQ234501.1 (Mutant 1%) (Abundance 6%), is almost fully recovered by viaDBG. The performance of metaSPAdes and SPAdes in this particular example was quite bad: they could only recover one of the 15 strains completely. Moreover, metaSPAdes did not retrieve any portion from 11 of them.

3.4 Efficiency analysis

We measured the runtime and peak memory usage required by all the algorithms when applied to each data set. All algorithms were given access to 32 cores in all experiments. Error correction, adding paired-end information, and polishing the paired-end information have been parallelised in viaDBG. The Supplementary Material includes an experiment evaluation of the effects of the number of cores used by each of the tools.

As shown in Table 2, SAVAGE needs much more time than the rest of the methods, ranging from 204.40 minutes in the fastest case to 352.98 minutes in the slowest one. This is caused by the computations made by SAVAGE during the overlap graph construction, which requires the enumeration of all approximate suffix-prefix overlaps among the reads. PEHaplo, despite following also an overlap graph approach, obtains much better execution times, as it removes a high percentage of repeated reads, thus, alleviating the construction of the graph. On the other hand, the time performance of the methods based on the de Bruijn graph is better: SPAdes and metaSPAdes were around 1.5 times faster than viaDBG. However, when the correction step is omitted, viaDBG outperforms both of them on the real data set, HIV-real. Results show that viaDBG worsens its time efficiency when including the error correction step, as the CPU times are around 4.5 times higher. This is an expected result, since the time complexity of the error correction performed by viaDBG is $O(n * m)$ where n is the number of reads and m is the maximum length of the reads.

PEHaplo is faster than viaDGB with correction step for HIV-real data set, but viaDBG obtains more accurate results. In the Supplementary Material we can also see that PEHaplo obtains slightly better time efficiency and also better accuracy than viaDBG for HCV-10.

We also measured the peak memory required by each tool. Among the de Bruijn methods, SPAdes and metaSPAdes require the highest memory resources, reaching 5.52 GB, whereas viaDBG requires at most 3.74 GB per execution. Only for ZIKV-15 data set, metaSPADES obtains lower memory consumption, but also much lower accuracy. On the other hand, if we consider the overlap methods, the memory requirements of SAVAGE are much higher, from 9.03 GB to 49.12 GB, depending on the file size, whereas PEHaplo requires for their worst tested case, HIV-5, 4.86 GB (8.99 GB if we consider HCV-10, as seen in the Supplementary Material).

3.5 Testing viaDBG limits

In this section, we will explore the algorithm bounds. We will follow the methodology used in the experimental evaluation of Baaijens *et al.* (2017), using 36 simulated data sets that vary their abundance and divergence.

Figure 5 shows the results obtained by viaDGB, SAVAGE and PEHaplo for each of these data sets in terms of percentage of retrieved genome and percentage of mismatches. In this experiment, in the case of SAVAGE, PEAR was applied over the input data sets. As we can see, viaDBG has a surprising behaviour with 10%, 5% and 2.5%, as it is able to retrieve almost the complete genome until the 1:50 abundance relation. Furthermore, on 1:50 relation, it is able to retrieve around 60% of the genome for the minor strain. Comparing our results with those achieved by SAVAGE and PEHaplo, we can see that viaDBG behaves better than either of them in these scenarios with higher differences of abundance rates. For example in the 1:50 case, neither SAVAGE nor PEHaplo are able to retrieve more than 10–20% of the minor strain, and in most cases they retrieve 0% of the minor strain. On the divergence bound behaviour, viaDBG's results decrease in comparison to SAVAGE, retrieving a bit less genome fraction when divergence is below 1%. A possible reason for this is the length of the processed k-mers. SAVAGE uses the full-length reads (>200 bp) and extends them, which produces much longer reads, thus more accuracy. On the other hand, viaDBG uses fixed k-mer length, which produces a slight loss in accuracy when divergence is below 1%. Nevertheless, it seems that both SAVAGE and viaDBG have a more robust behaviour than PEHaplo, which suffers when divergence is below 0.75%. In conclusion, viaDBG can properly handle different ranges of divergence levels and of relative abundances, especially for extreme differences in the abundance ratio.

3.6 Real data sets with unknown target genome

In this section, we show the results obtained for a real virus sample from patients infected by the Asian lineage Zika virus. To evaluate the results, we use as references the complete genome sequence of the Asian lineage Zika virus (KU681081.3). The result for the Hepatitis C virus can be found in the Supplementary Material.

We obtain 10 contigs above 1000 bp covering 9578 out of 10677 bases, with a N50 of 1975 bp and a largest contig of 2445 bp. Additionally, 17809 bases were aligned, thus it is obvious that more than one strain is contained in the sample. According to these results, it seems that there are two different, but highly similar strains, in the sample. Besides, in our analysis, we have not discovered any local misassembly, which means that there is no contig that does not align with the reference at some point.

4 Conclusion

We present viaDBG, a time- and memory-efficient *de novo* multi-assembler for viral quasispecies. Viral samples generally contain several haplotypes which have evolved from the same genome through multiple mutations and recombination events. Additionally, not all viral genomes within the sample have exactly the same frequency, namely each viral genome has its own level of abundance. Experimental results have shown that general purpose and metagenomic assemblers, such as SPAdes and metaSPAdes, are not able to retrieve the viral genomes in the sample. This motivates the research on new specific tools that can overcome all these limitations.

Our experimental evaluation shows that viaDBG is able to get competitive, sometimes even better, results in comparison to state-of-the-art *de novo* viral quasispecies assemblers, such as SAVAGE and PEHaplo. Furthermore, the runtime of viaDBG is much lower than SAVAGE and also to PEHaplo in most cases, and its memory usage is also lower than its counterparts, making viaDBG an attractive alternative. One of the main drawbacks of PEHaplo is that its behaviour is highly dependent on the parameter configuration, which is not easy to determine for each particular data set. Furthermore, in some extremely complex cases, such as 15 ZIKV strains where genomes are extremely close and abundance is extremely low, viaDBG is able to retrieve information for the whole set of strains and the overall genome fraction is higher than for other tools. Despite of these successful results, our method shows a weaker behaviour than SAVAGE for data sets with extreme divergence ratios.

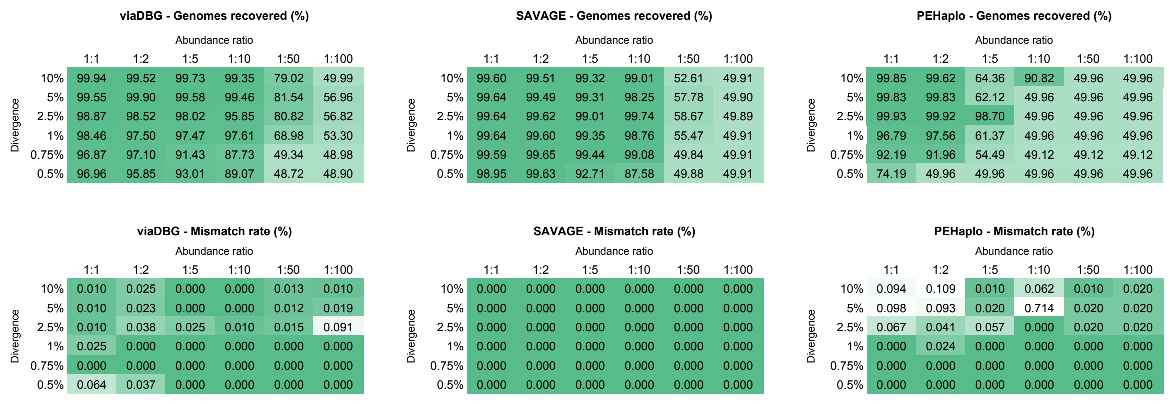
The main reasons for the good performance of viaDBG are the error correction step and the systematic use of the paired-end information. On the one hand, error correction enables the usage of extremely large k-mers (120-mers), as the veracity of the k-mers is improved due to the adjustment of their frequency distribution. Additionally, a side effect of the correction is that it reduces the possibility of having wrong pairs for genomic k-mers and vice versa. On the other hand, the cliques retrieved by using the paired-end information for every pair of k-mers allow us to change the original de Bruijn graph into a much less tangled graph. Applying paired information as a post-processing step is more restrictive than adding the information directly in the graph. When used during the post-processing step, only reads that align entirely with one contig are going to be used, whereas all reads with genomic information can improve the results when they are considered during the graph construction. Therefore, there is always more information when using paired k-mers than when using the overlap between reads and contigs.

The main advantage of viaDBG is its efficiency, both in terms of execution time and memory usage. Our method benefits from the better efficiency of de Bruijn graph approaches, which avoids computing overlaps between all reads. Despite the computations of the paired information, viaDBG has proven to be much faster than overlap based methods.

As a future work, we plan to reduce the memory footprint of viaDBG by taking full advantage of compacted de Bruijn graphs. This will also allow us to study the suitability of our approach for metagenomics assembling, which is a more demanding task. Another line of improvement is to enhance the current parallelisation of viaDBG by taking into account some relevant issues such us disk accesses, thread synchronization, and data interchanges. In parallel, we will consider the possibility of integrating our approach with Virus-VG (Baaijens *et al.*, 2019) to produce larger contigs.

Acknowledgements
We want to thank J. Baaijens and J. Chen for the help in the execution and in the search for the right parameters on SAVAGE and PEHaplo respectively.

Funding
This research has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie [grant agreement No 690941]; Ministerio de Ciencia, Innovación y Universidades [TIN2016-78011-C4-1-R; TIN2016-77158-C4-3-R; FPU17/02742]; Xunta de Galicia [ED431C 2017/58; ED431G/01; IN848D-2017-2350417; IN852A 2018/14]; and from Academy of Finland [308030, 314170, and 323233].



viaDBG - Genomes recovered (%)

Divergence \ Abundance ratio	1:1	1:2	1:5	1:10	1:50	1:100
10%	99.94	99.52	99.73	99.35	79.02	49.99
5%	99.55	99.90	99.58	99.46	81.54	56.96
2.5%	98.87	98.52	98.02	95.85	80.82	56.82
1%	98.46	97.50	97.47	97.61	68.98	53.30
0.75%	96.87	97.10	91.43	87.73	49.34	48.98
0.5%	96.96	95.85	93.01	89.07	48.72	48.90

SAVAGE - Genomes recovered (%)

Divergence \ Abundance ratio	1:1	1:2	1:5	1:10	1:50	1:100
10%	99.60	99.51	99.32	99.01	52.61	49.91
5%	99.64	99.49	99.31	98.25	57.78	49.90
2.5%	99.64	99.62	99.01	99.74	58.67	49.89
1%	99.64	99.60	99.35	98.76	55.47	49.91
0.75%	99.59	99.65	99.44	99.08	49.84	49.91
0.5%	98.95	99.63	92.71	87.58	49.88	49.91

PEHaplo - Genomes recovered (%)

Divergence \ Abundance ratio	1:1	1:2	1:5	1:10	1:50	1:100
10%	99.85	99.62	64.36	90.82	49.96	49.96
5%	99.83	99.83	62.12	49.96	49.96	49.96
2.5%	99.93	99.92	98.70	49.96	49.96	49.96
1%	96.79	97.56	61.37	49.96	49.96	49.96
0.75%	92.19	91.96	54.49	49.12	49.12	49.12
0.5%	74.19	49.96	49.96	49.96	49.96	49.96

viaDBG - Mismatch rate (%)

Divergence \ Abundance ratio	1:1	1:2	1:5	1:10	1:50	1:100
10%	0.010	0.025	0.000	0.000	0.013	0.010
5%	0.010	0.023	0.000	0.000	0.012	0.019
2.5%	0.010	0.038	0.025	0.010	0.015	0.091
1%	0.025	0.000	0.000	0.000	0.000	0.000
0.75%	0.000	0.000	0.000	0.000	0.000	0.000
0.5%	0.064	0.037	0.000	0.000	0.000	0.000

SAVAGE - Mismatch rate (%)

Divergence \ Abundance ratio	1:1	1:2	1:5	1:10	1:50	1:100
10%	0.000	0.000	0.000	0.000	0.000	0.000
5%	0.000	0.000	0.000	0.000	0.000	0.000
2.5%	0.000	0.000	0.000	0.000	0.000	0.000
1%	0.000	0.000	0.000	0.000	0.000	0.000
0.75%	0.000	0.000	0.000	0.000	0.000	0.000
0.5%	0.000	0.000	0.000	0.000	0.000	0.000

PEHaplo - Mismatch rate (%)

Divergence \ Abundance ratio	1:1	1:2	1:5	1:10	1:50	1:100
10%	0.094	0.109	0.010	0.062	0.010	0.020
5%	0.098	0.093	0.020	0.714	0.020	0.020
2.5%	0.067	0.041	0.057	0.000	0.020	0.020
1%	0.000	0.024	0.000	0.000	0.000	0.000
0.75%	0.000	0.000	0.000	0.000	0.000	0.000
0.5%	0.000	0.000	0.000	0.000	0.000	0.000

Fig. 5. Performance of viaDBG, SAVAGE, and PEHaplo for different divergence and abundance ratios.

References

Ahn, S. and Vikalo, H. (2018). aBayesQR: a bayesian method for reconstruction of viral populations characterized by low diversity. *Journal of Computational Biology*, **25**(7), 637–648. PMID: 29480740.

Baaijens, J. A., Aabidine, A. Z. E., Rivals, E., and Schönhuth, A. (2017). De novo assembly of viral quasispecies using overlap graphs. *Genome Research*, **27**, 835–848.

Baaijens, J. A., Van der Roest, B., Köster, J., Stougie, L., and Schönhuth, A. (2019). Full-length de novo viral quasispecies assembly through variation graph construction. *Bioinformatics*. btz443.

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A., Dvorkin, M., Kulikov, A., Lesin, V., Nikolenko, S., Pham, S., Prjibelski, A., Pyshkin, A., Sirotkin, A., Vyahhi, N., Tesler, G., Alekseyev, M., and Pevzner, P. (2012). SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, **19**(5), 455–477.

Barik, S., Das, S., and Vikalo, H. (2018). QSdpR: viral quasispecies reconstruction via correlation clustering. *Genomics*, **110**(6), 375 – 381.

Chen, J., Zhao, Y., and Sun, Y. (2018). *De novo* haplotype reconstruction in viral quasispecies using paired-end read guided path finding. *Bioinformatics*, **34**(17), 2927–2935.

Domingo, E., Sheldon, J., and Perales, C. (2012). Viral quasispecies evolution. *Microbiology and Molecular Biology Reviews*, **76**(2), 159–216.

Dudley, D. M. *et al.* (2016). A rhesus macaque model of Asian-lineage Zika virus infection. *Nature Communications*, **7**, 12204.

Duffy, S., Shackelton, L. A., and Holmes, E. C. (2008). Rates of evolutionary change in viruses: patterns and determinants. *Nature Reviews Genetics*, **9**, 267–276.

Giallonardo, F. D., Töpfer, A., Rey, M., Prabhakaran, S., Duport, Y., C, C. L., Schmutz, S., Campbell, N. K., Joos, B., Lecca, M. R., Patrignani, A., Däumler, M., Beisel, C., Rusert, P., Trkola, A., Günthard, H. F., Roth, V., Beerenwinkel, N., and Metzner, K. J. (2014). Full-length haplotype reconstruction to infer the structure of heterogeneous virus populations. *Nucleic Acids Res*, **42**(14), e115.

Holmes, E. C. (2009). *The Evolution and Emergence of RNA Viruses*. Oxford University Press.

Jayasundara, D., Saeed, I., Maheswararajah, S., Chang, B., Tang, S.-L., and Halgamuge, S. K. (2015). ViQuaS: an improved reconstruction pipeline for viral quasispecies spectra generated by next-generation sequencing. *Bioinformatics*, **31**(6), 886–896.

Knyazev, S., Tsyvina, V., Melnyk, A., Artyomenko, A., Malygina, T., Porozov, Y. B., Campbell, E., Switzer, W. M., Skums, P., and Zelikovsky, A. (2019). CliqueSNV: scalable reconstruction of intra-host viral populations from ngs reads. *bioRxiv*.

Malhotra, R., Wu, M. M. S., Rodrigo, A., Poss, M., and Acharya, R. (2015). Maximum likelihood de novo reconstruction of viral populations using paired end sequencing data. *arXiv e-prints*, page arXiv:1502.04239.

Martin, M. (2011). Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal*, **17**(1), 10–12.

Medvedev, P., Pham, S., Chaisson, M., Tesler, G., and Pevzner, P. (2011). Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, **18**(11), 1625–1634.

Mikheenko, A., Saveliev, V., and Gurevich, A. (2016). MetaQUAST: evaluation of metagenome assemblies. *Bioinformatics*, **32**(7), 1088–1090.

Nagarajan, N. and Pop, M. (2013). Sequence assembly demystified. *Nature Review Genetics*, **14**, 157–167.

Nurk, S., Meleshko, D., Korobeynikov, A., and Pevzner, P. (2017). metaSPAdes: a new versatile metagenomic assembler. *Genome Research*, **27**, 824–834.

Posada-Cespedes, S., Seifert, D., and Beerenwinkel, N. (2017). Recent advances in inferring viral diversity from high-throughput sequencing data. *Virus Research*, **239**, 17–32.

Prabhakaran, S., Rey, M., Zagordi, O., Beerenwinkel, N., and Roth, V. (2014). HIV haplotype inference using a propagating Dirichlet process mixture model. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **11**(1), 182–191.

Prosperi, M. C. F. and Salemi, M. (2011). QuRe: software for viral quasispecies reconstruction from next-generation sequencing data. *Bioinformatics*, **28**(1), 132–133.

Salmela, L. and Rivals, E. (2014). LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, **30**(24), 3506–3514.

Töpfer, A., Zagordi, O., Prabhakaran, S., Roth, V., Halperin, E., and Beerenwinkel, N. (2013). Probabilistic inference of viral quasispecies subject to recombination. *Journal of Computational Biology*, **20**(2), 113–123.

Töpfer, A., Marschall, T., Bull, R. A., Luciani, F., Schönhuth, A., and Beerenwinkel, N. (2014). Viral quasispecies assembly via maximal clique enumeration. *PLOS Computational Biology*, **10**(3), e1003515.

Zagordi, O., Bhattacharya, A., Eriksson, N., and Beerenwinkel, N. (2011). ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data. *BMC Bioinformatics*, **12**, 119.

Zhang, J., Kobert, K., Flouri, T., and Stamatakis, A. (2014). PEAR: a fast and accurate Illumina Paired-End reAd mergeR. *Bioinformatics*, **30**(5), 614–620.

Supplementary Material

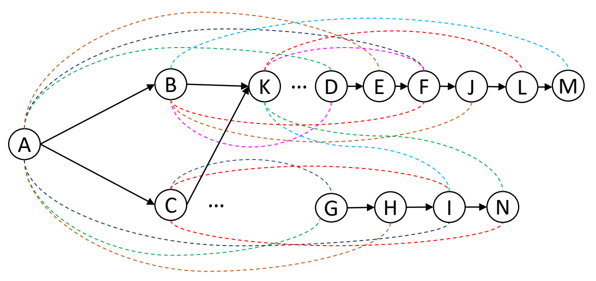


Fig. 1. DBG with the paired-end information.

Contents

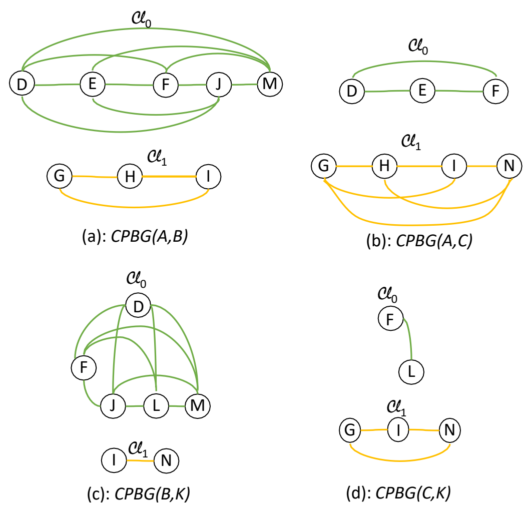


Fig. 2. The four different clique graphs.

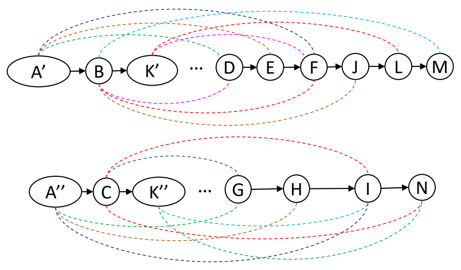


Fig. 3. The final form of the graph.

1 Algorithm for obtaining the haplotypes

Algorithm 1 shows the pseudocode of the algorithm for obtaining the haplotypes, described in Section 2.4.4 of the main paper, that is, steps 2.(e), 2.(f), and 2.(g) of the Figure 1 of the main paper.

Figure 1 shows a portion of a DBG used to illustrate this process. Solid black arrows are the edges of DBG, whereas the coloured dotted lines show the pair-end information of each node. .

The algorithm receives as input the DBG obtained through the steps 2.(a) until 2.(d) of the Figure 1 of the main paper. In Line 4, for each pair of adjacent nodes (A, B) of the DBG, a CPBG (*CPBG(A,B)*) is built. Lines 6 and 7 create the nodes of *CPBG(A,B)*. Lines 8–10 add the edges of the CPBG.

Figure 2(a) shows *CPBG(A,B)*.[1] It is composed of the nodes in *P(A)={D, E, F, G, H, I}* and *P(B)={M, D, F, J}*. There is an edge between two nodes if there is a path in the DBG of length smaller than 2Δ. In this case, function $reach_{DBG}(A, B)$ returns *true*.

Line 13 obtains the set of maximal cliques in the CPBG (denoted as $\mathcal{SC}$). This algorithm is shown in Section 2 of this Supplementary Material.

Figure 2 shows the four different CPBGs obtained for the DBG shown in Figure 1. Each clique in the graphs is highlighted by its own color.

Line 14 polishes the cliques in $\mathcal{SC}$. For example, in *CPBG(A,B)*, the yellow clique will be discarded because it only has nodes from *P(A)*. This step avoids having short tips (1 bp) and having several contigs which lead to exactly the same strain.

Lines 15–28 create the nodes of the new DBG. Line 15 is a loop that processes all cliques of the treated pair. If the processed clique has nodes of $P(A)$ and $P(B)$, then that pair is added to the new DBG, otherwise it is discarded (Line 16).

In Figure 2(a), only $\mathcal{C}\ell_0$ passes the polish step. Therefore the pair (A,B) is added to the output (see Figure 3): A with paired information $P(A) \cap \mathcal{C}\ell_0 =$*{D, E, F}* and B with paired information $P(B) \cap \mathcal{C}\ell_0 =$ *{D, F, J}*.

Processing $\mathcal{C}\ell_1$ of *CPBG(A,C)* (see Figure 2(b)) produces the output of A and C. However, that A is *different*[2] from that obtained from *CPBG(A,B)*, and thus, to differentiate them, we use A' for the one obtained from *CPBG(A,B)* and A'' for the one obtained from *CPBG(A,C)*. Continuing the process, we obtain the new DBG of Figure 3.

2 Obtaining the maximal cliques in the CPBG

Classic algorithms to find cliques in undirected graphs, such as those of Johnson *et al.* (1988) or Tomita *et al.* (2006), have a considerable computational cost, for example, $O(3^{n/3})$ in the case of Tomita *et al.*

[1] We do not use representative k-mers to facilitate the understanding of the example.

[2] In the sense that they belong to different strains.

Algorithm 1 Obtain Haplotypes (DBG)

```
1: let $new\_DBG$ be a DBG
2: let $new\_DBG_{nodes} = \emptyset$
3: let $new\_DBG_{edges} = \emptyset$
4: for all $(A, B)$ a pair of adjacent nodes of the $DBG$ do {Builds the CPBG}
5:   let $CPBG(A, B)_{edges} = \emptyset$
6:   let $CPBG(A, B)_{nodes} = P(A)$ {Adds the representative $k$-mers in P(A)}
7:   let $CPBG(A, B)_{nodes} = CPBG(A, B)_{nodes} \cup P(B)$ {Adds the representative $k$-mers in P(B)}
8:   for all $(U, V)$ nodes in $CPBG(A, B)_{nodes}$ do
9:     if $reach_{DBG}(U, V)$ then
10:      $CPBG(A, B)_{edges} = CPBG(A, B)_{edges} \cup \{(U, V)\}$
11:    end if
12:  end for
13:  let $\mathcal{SC} = \{\mathcal{C}\ell_0, \mathcal{C}\ell_1, ..., \mathcal{C}\ell_n\}$ the maximal cliques in $CPBG(A, B)$
14:  polish $(\mathcal{SC})$
15:  for all $\mathcal{C}\ell_i \in \mathcal{SC}$ do
16:    if $\mathcal{C}\ell_i \cap P(A) \neq \emptyset$ and $\mathcal{C}\ell_i \cap P(B) \neq \emptyset$ then
17:      let $new\_DBG_{nodes} = new\_DBG_{nodes} \cup \{A_{P(A)\cap\mathcal{C}\ell_i}\}$ {Adds version $A$ with paired information $P(A) \cap \mathcal{C}\ell_i$}
18:      let $new\_DBG_{nodes} = new\_DBG_{nodes} \cup \{B_{P(B)\cap\mathcal{C}\ell_i}\}$ {Adds version $B$ with paired information $P(B) \cap \mathcal{C}\ell_i$}
19:      let $new\_DBG_{edges} = new\_DBG_{edges} \cup \{(A_{P(A)\cap\mathcal{C}\ell_i}, B_{P(B)\cap\mathcal{C}\ell_i})\}$ {Both nodes are connected}
20:    end if
21:  end for
22: end for
23: search unitigs in $new\_DBG$
```

Algorithm 2 Obtain maximal Cliques $(DBG, CPBG(A, B))$

```
1: for all Node $n$ in $CPBG(A, B)$ do
2:   for all Edge $e$ of $DBG$ that reaches $n$ do
3:     let $n.weight$ =+ $e.weight$
4:   end for
5: end for
6: for all Node $n$ in $CPBG(A, B)$ do
7:   for all Node $n'$ in $CPBG(A, B)$ adjacent to $n$ do
8:     let $n.weight$ =+ $n'.weight$
9:   end for
10: end for
11: let $i = 0$
12: let $\mathcal{C}\ell_i = \emptyset$
13: repeat
14:   let=$n$ be the node of $CPBG(A, B)$ with the highest value of $degree \times n.weight$
15:   repeat
16:     let $\mathcal{C}\ell_i = \mathcal{C}\ell_i \cup n$
17:     let $n_{used} = true$
18:     let $n' = n$
19:     let=$n$ be the node of $CPBG(A, B)$ adjacent to $n'$ and also connected to all nodes in $c_i$ with the highest value $degree \times n.weight$ and not in $c_i$
20:   until $n = \emptyset$
21:   let $lowest$ be the lowest weight of all nodes in $c_i$
22:   for all Node $n$ in $\mathcal{C}\ell_i$ do
23:     let $n.weight$ =- $lowest$
24:     if $n.weight$==0 then
25:       let $n.weight = 1$
26:     end if
27:   end for
28:   let i=i+1
29:   let $\mathcal{C}\ell_i = \emptyset$
30: until All nodes in $CPBG(A, B)$ are used
31: return $\mathcal{C}\ell_0, \mathcal{C}\ell_1, \ldots, \mathcal{C}\ell_v$
```

(2006). Therefore, instead, we use a faster heuristic method shown in Algorithm 2, which is based on the work by Pattabiraman *et al.* (2015).

It receives as input the DBG and the CPBG of a given pair of nodes. Observe in Figure 4, that the edges of the DBG corresponding to paired information are now labeled with the number of paired reads that contain the corresponding pair of k-mers. That frequency is used to determine which cliques correspond to real strains.

The first loop of Lines 1–4 enriches the nodes of the CPBG with a number which results from adding the weight of all edges which reach that node in the DBG. Figure 5(a) shows the result of this process for the *CPBG(A,B)* of our example in Figure 4. For example, observe the node D, its weight (18) is obtained by adding the weight of the edge of the DBG connecting A and D, with weight 10, and that connecting B and D, with weight 8.

The next loop in Lines 6–10 adds more weight to each node of the CPBG, specifically, the weight of all its adjacent nodes. Figure 5(b) shows the result. The weight of D is 45, resulting from adding its weight (18), and those of its adjacent nodes (J (8), F (9), and E (10)).

The loop of Lines 13–29 is the main part of the algorithm. Line 14 selects the node with the highest value resulting from multiplying its degree (number of adjacent nodes) and its weight. In our example, that node is F.

Then, the loop of Lines 15–20 builds the clique containing that node. In our example, first F is added to the clique and marked as "used". Then, line 19 obtains the adjacent node of F with the highest value resulting from multiplying its degree and its weight. In our example, all nodes are adjacent to F, but E and J are those with the highest value $(46 * 4)$. So, the process continues adding, let say, E and then J and marking them as "used". From J, the adjacent node with highest value is D, which is also added to the clique. After processing D, all its adjacent nodes are already in the clique, and thus the process ends.

Next, Lines 21–27 decrease the weight of all the nodes of the recently created clique by subtracting the weight of the node of the clique with the lowest value, except if the result of that subtraction is 0, which is changed to 1. The resulting clique of this process is shown in Figure 5(c) highlighted in yellow. Observe that its nodes remain now with a low weight, then it is unlikely that these nodes will be part of subsequent cliques. The idea is that we have concluded that those nodes correspond to a given strain, it is unlikely that they would be part of another different strain.

The process continues until all nodes have been marked as "used". In our example, another clique is created, as shown in Figure 5(d), but this clique is later removed by the polishing process, as explained in the main manuscript.

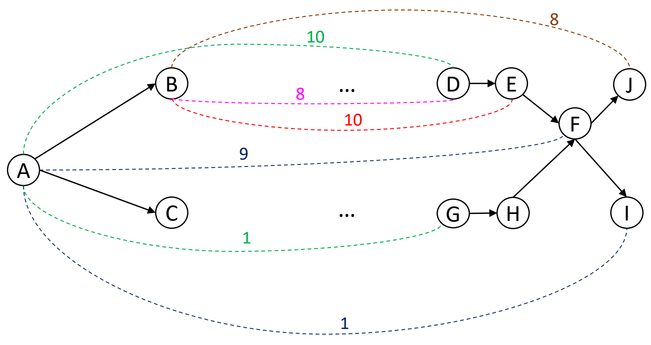


Fig. 4. DBG with the frequency of appearance of paired $k-$mers.

3 Parameter selection

The most important parameters of our method are *k-mer solid threshold* and the variance of the insert size which we denote by Δ. The Δ value can be configured using a higher value than needed without endangering the assembly results. Nevertheless, if some further information is known, then using a more precise Δ will provide better results avoiding some false cliques that may appear. On the other hand, the *k-mer solid threshold* is automatically selected by using the histogram of the k-mers frequencies, where a change in the trend of the frequency count intuitively means that from there on k-mers having higher frequencies are genomic. For better identifying this point of change, we use a window of size N.

We have conducted several experiments for analysing the effect of this window size in the results. We obtained that the threshold selection for the simulated data is quite stable, whereas it is much more variable for the real data set HIV-real. A possible explanation is given by Figure 6, which shows two k-mer frequency histograms: one for the real data set HIV-real and the other for the simulated data set HIV-5. Nevertheless, results in terms of genome fraction retrieved have been consistent throughout the whole evaluation. This information is provided in Figures 7 and 8. In Figure 7, we can see that for HIV-real, the threshold suggested by our algorithm increases when the windows size grows, whereas for the simulated data sets, the threshold is practically stable. On the other hand, we can see in Figure 8 that the windows size does not affect the percentage of the retrieved genome for any data set.

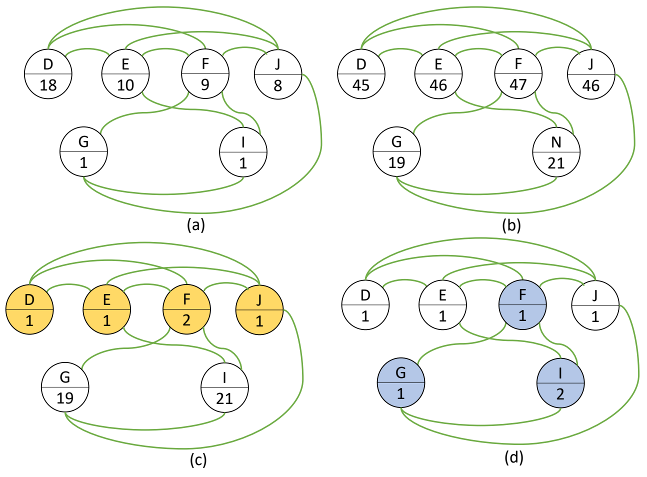


Fig. 5. Obtaining the cliques of the *CPBG(A,B)* of the DBG of Figure 4.

4 Description of Zika virus simulated data sets (ZIKV-3 and ZIKV-15)

SAVAGE was assayed by using a 15-strain ZIKV data set. Unfortunately, ground truth criteria, namely reference, was not available. Therefore, we simulated our own references and the data set.

Twelve extra references were produced from only 3 strains, all of African lineage: one from Uganda (accession HQ234498), one from Nigeria (accession HQ234500), and one from Senegal (accession HQ234501). For each reference 4 extra sequences were build by inserting 1%, 1%, 2% and 2% mutations to the reference.

The data set was simulated by using the whole set of 15 references with abundances from 1% to 13% at most.

5 Complete results comparison - overall performance

We include here the complete results of the benchmarking performed in Section 3 of the main paper. Thus, Table 1 contains the results for all the data sets, included ZIKV-3 and HCV-10, which were omitted in Table 2 of the main paper. We also include the values for the number of contigs larger than 500 bp, the length of the largest contig, the percentage of indels, N-rate, and total user CPU time. We measured peak memory usage using gnu time commnad (with option -v), and measured time performance using perf profiler (command perf stat -d), which reports both total CPU user time and elapsed time.

5.1 Analysis of the parallelisation

We conducted an experiment varying the number of cores available to the methods to see how the number of cores affects the running times and the

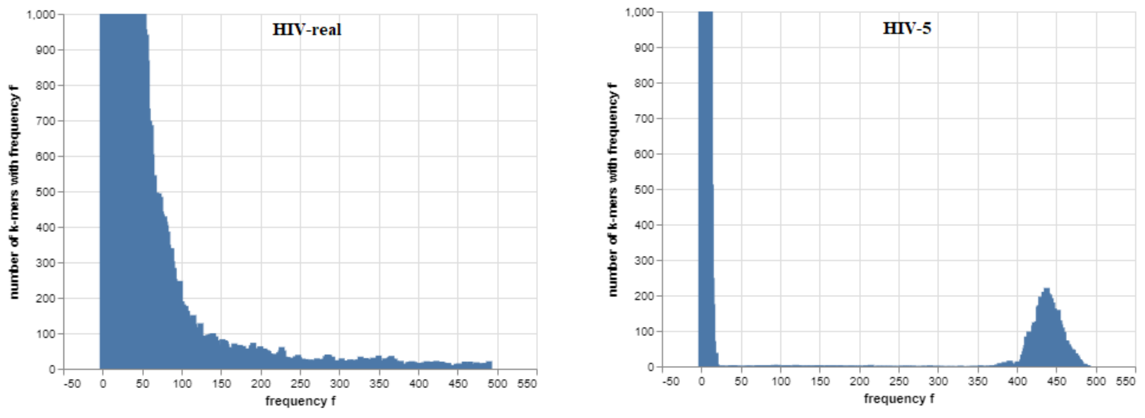


Fig. 6. HIV-real vs HIV-5 frequency histograms.

Table 1. Assembly results per method on the benchmarking data sets when ground truth is known.

data set	method	contigs >500	% genome	N50	largest contig	mis-assemb.	% mis-matches	% indels	% N-Rate	elapsed time (min)	CPU user time (min)	peak mem (GB)
HIV-real	viaDBG - without correction	88	87.25%	1813	8596	0	0.197	0.285	0.000	4.48	17.24	3.74
	viaDBG - with correction	57	89.53%	1986	8966	0	0.204	0.240	0.000	20.01	389.13	3.74
	SAVAGE	459	91.79%	611	2511	0	0.684	0.149	0.104	218.30	4803.06	49.12
	PEHaplo - with polishing	35	87.96%	2995	8674	0	3.521	0.245	0.000	12.74	101.89	3.74
	PEHaplo - without polishing	31	91.43%	1262	6383	0	0.074	0.083	0.000	7.56	50.47	3.74
	SPAdes	1	20.15%	660	2952	1	2.091	0.089	0.000	12.74	99.63	5.52
	metaSPAdes	16	83.10%	1432	2986	3	9.291	0.405	0.000	9.06	111.43	4.29
HIV-5	viaDBG	45	97.50%	8046	9667	2	0.151	0.008	0.000	5.01	63.56	2.89
	SAVAGE	18	97.69%	3305	9645	4	0.120	0.004	0.000	204.40	3618.10	26.11
	PEHaplo	7	78.59%	9328	9656	2	0.690	0.037	0.000	23.93	68.58	4.86
	SPAdes	22	90.91%	5097	9557	2	0.051	0.002	0.000	3.31	25.89	4.12
	metaSPAdes	8	35.87%	6385	6561	6	5.322	0.104	0.000	3.86	51.52	2.99
ZIKV-3	viaDBG	10	99.76%	10203	10267	0	0.000	0.000	0.000	7.56	62.10	3.66
	SAVAGE	3	99.77%	10243	10258	0	0.003	0.000	0.003	332.15	6527.90	42.37
	PEHaplo	2	99.89%	10247	10269	0	0.000	0.000	0.000	20.11	68.19	4.40
	SPAdes	3	99.56%	9851	10269	0	0.000	0.000	0.000	4.05	33.05	4.59
	metaSPAdes	6	33.34%	2890	8675	0	1.919	0.009	0.000	4.96	58.69	3.33
ZIKV-15	viaDBG	185	86.06%	1759	9483	0	0.002	0.000	0.000	18.26	82.22	3.71
	SAVAGE	231	82.72%	1632	10199	0	0.002	0.000	0.002	352.98	8329.14	9.03
	PEHaplo	-	-	-	-	-	-	-	-	-	-	-
	SPAdes	247	38.97%	2063	10251	0	0.147	0.000	0.000	6.17	35.34	4.42
	metaSPAdes	11	16.03%	3863	5261	0	2.273	0.264	0.000	4.49	57.98	3.19
HCV-10	viaDBG	27	97.72%	8934	9293	0	0.005	0.000	0.000	13.03	69.06	2.81
	SAVAGE	20	99.33%	9204	9290	0	0.0975	0.000	1.043	50.01	451.63	26.13
	PEHaplo	10	99.66%	9297	9311	0	0.032	0.000	0.000	29.01	64.79	8.99
	SPAdes	26	90.59%	8690	9311	0	0.002	0.000	0.000	4.10	26.79	4.09
	metaSPAdes	42	49.37%	2742	3475	0	4.534	0.000	0.000	3.73	49.86	2.97

memory usage. We used HIV-real data set with the same configuration for each tool as the previous experiments. Table 2 shows that only SAVAGE approaches an ideal speedup, as times are practically divided by 2 when doubling the number of cores. In the case of viaDBG, using 16 cores instead of 8 only implies an improvement of 2%, whereas the usage of 32 cores yields a speedup of 1.78. Still, that value is around the same as that obtained by metaSPAdes and better than the rest, except for SAVAGE.

In Table 3, we can observe that memory usage grows for all methods when increasing the number of cores. Again, viaDBG obtains similar results using 8 and 16 cores, but the memory usage growth is worse for 32 cores. This increase in memory consumption is moderate in the case of PEHaplo, which is the tool obtaining the best result for 32 cores, but larger for metaSPAdes and SAVAGE.

From the results, we can see that SAVAGE is the tool that best takes advantage of parallelisation, as its running time considerably decreases with the number of cores, at the expense of greater memory requirements. In any case, those time results are still much higher than those obtained by the rest of the techniques. viaDBG parallelisation obtains a slight speedup at the expense of a moderate growth in memory consumption.

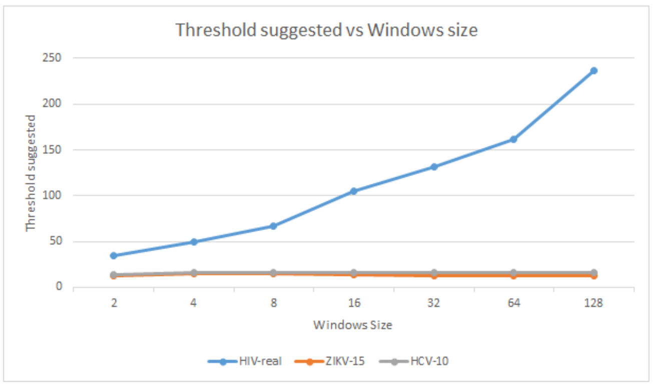


Fig. 7. Threshold value suggested by our approach, varying the windows size.

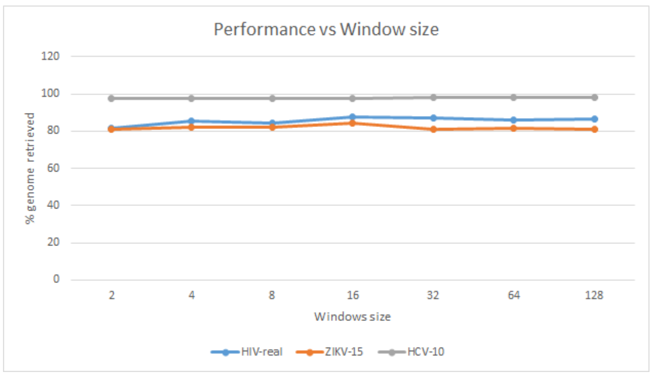


Fig. 8. Genome fraction retrieved, varying the windows size.

Table 2. Running times in minutes, varying the number of cores.

# of cores	viaDBG	SPAdes	metaSPAdes	PeHaplo	SAVAGE
8	5.96	15.51	15.73	11.00	1255.91
16	5.82	12.83	9.72	11.18	468.58
32	3.34	12.74	9.06	7.56	218.30

Table 3. Memory usage in Gigabytes, varying the number of cores.

# of cores	viaDBG	SPAdes	metaSPAdes	PeHaplo	SAVAGE
8	2.85	4.97	6.53	4.12	7.26
16	2.96	7.28	11.57	4.00	10.25
32	6.69	11.35	19.03	5.50	20.27

6 Assembly of real data sets with unknown target genome

6.1 BAC clones in the Zika virus sample

Baaijens *et al.* (2017) discovered human BAC clones within the same real Zika virus data set that we have also analysed. In our results we did not find these. This could be caused by two possible explanations. On the one hand, it is probable that BAC information was not complete, creating several isolated connected components that were removed during the polishing step of the graph. On the other hand, results can differ due to the preprocessing step, which is not exactly the same as the one used by SAVAGE. In fact, results obtained by SAVAGE method with our preprocessing step did not show BAC clones either.

6.2 Hepatitis C Human Sample

Here, we show the results of the Australian human patient infected with Hepatitis C virus. For ground truth, we will use the complete genome of Hepatitis C virus (NC_004102.1).

Obviously, the results are much more non-specific than in the Zika sample. This is probably because of the Australian warm environment, the time that the virus spent in the carrier, plus the initial number of strains was as well unknown.

7 Commands executed

We run the tools over the same data sets, but customising the configuration with the best one for each data set. In the case of PEHaplo, and following authors' indications, input was only trimmed but PEAR was not used.

7.1 Specialized assembly tools

- ***viaDBG***: *Version 1.0*
  Current version of viaDBG only allows dsk as counter. Furthermore, preprocessing included removing duplicated reads, as this boosts efficiency with no accuracy impact.

  - ./bin/viaDBG -p ../PairEndDir/ -o Output -u ../OutputUnitig -k 1...192 -c dsk -n -t 1 --postprocess
  - ./bin/viaDBG -s SingleEndFile -p ../PairEndDir/ -o Output -u ../OutputUnitig -k 1...192 -c dsk -n -t 1 --postprocess

- ***SAVAGE***: *Version: 0.4.0*
  Although last version of SAVAGE has made the selection of minimum overlap automatic, it is recommended to use the value 150.

  - python savage.py -p1 forward.fastq -p2 reverse.fastq -t 32 --split 30
  - python savage.py -s single-end.fastq -p1 forward.fastq -p2 reverse.fastq -t 32 --split 30

- ***PEHaplo***: *Version 1.0*
  Data sets for PEHaplo were modified to fit the software requirements:

  - Reads ids must be all different.
  - Reads names must end with /1.
  - Only fasta format is allowed.

  - python pehaplo.py -f1 forward.fasta -f2 reverse.fasta -l 210 -l1 220 -correct yes -n 3 -r 250 -F 450 -t 32

7.2 Generic assembly tools

- ***SPAdes***: *Version 3.13.1*
  SPAdes has automated the selection of multiple parameters such as k-mer size. Therefore, we relied on SPAdes for the parameters selection.

  - python spades.py -s pear.assembled.fastq -1 forward.fastq -2 reverse.fastq -t 32

- ***metaSPAdes***: *Version 3.13.1*
  metaSPAdes has also automated the selection of multiple parameters thus we again relied on metaSPAdes parameters selection.

  - python metaspades.py -s assembled.pear.fastq -1 forward.fastq -2 reverse.fastq -t 32

7.3 PEAR

- ./pear -f forward.fasta -r reverse.fasta -o output_preffix

7.4 Divergence-Abundance Comparison

- ***viaDBG***: viaDBG needs to use longer k-mers when the divergence ratio decreases. Abundance is not managed by any parameter.

  - Divergence above 1%: ./bin/viaDBG -p exp_0.1_1/ ../OutputDir/ -u ../OutputUnitigs -k 120 -c dsk -n -t 32
  - Divergence below 1%: ./bin/viaDBG -p exp_0.1_1/ ../OutputDir/ -u ../OutputUnitigs -k 180 -c dsk -n -t 32

- ***SAVAGE***:

  - python savage.py -p1 forward.fastq -p2 reverse.fastq -t 32 –split 1

- ***PEHaplo***:

  - Relative abundance 50%: python pehaplo.py -f1 forward.fasta -f2 read2.fasta -l 210 -l1 220 -correct yes -n 3 -r 250 -F 450 -t 32 -std 150
  - Relative abundance 33%: python pehaplo.py -f1 forward.fasta -f2 read2.fasta -l 210 -l1 220 -correct yes -n 2 -r 250 -F 450 -t 32 -std 150
  - Relative abundance below 10%: python pehaplo.py -f1 forward.fasta -f2 read2.fasta -l 210 -l1 220 -correct yes -n 1 -r 250 -F 450 -t 32 -std 150
  - Low divergence ratio: python pehaplo.py -f1 forward.fasta -f2 read2.fasta -l 170 -l1 200 -correct yes -n 1 -r 250 -F 450 -t 32 -std 150

References

Baaijens, J. A., Aabidine, A. Z. E., Rivals, E., and Schönhuth, A. (2017). De novo assembly of viral quasispecies using overlap graphs. *Genome Research*, **27**, 835–848.

Johnson, D. S., Yannakakis, M., and Papadimitriou, C. H. (1988). On generating all maximal independent sets. *Information Processing Letters*, **27**(3), 119 – 123.

Pattabiraman, B., Patwary, M. M. A., Gebremedhin, A. H., keng Liao, W., and Choudhary, A. (2015). Fast algorithms for the maximum clique problem on massive graphs with applications to overlapping community detection. *Internet Mathematics*, **11**(4–5), 421–448.

Tomita, E., Tanaka, A., and Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, **363**(1), 28 – 42. Computing and Combinatorics.