Simplitigs as an efficient and scalable representation of de Bruijn graphs

Karel Břinda[1,2,*], Michael Baym[1], and Gregory Kucherov[3,4]

1 Department of Biomedical Informatics, Harvard Medical School, Boston, USA

2 Center for Communicable Disease Dynamic, Department of Epidemiology, Harvard T.H. Chan School of Public Health, Boston, USA

3 CNRS/LIGM Univ Gustave Eiffel, Marne-la-Vallée, France

4 Skolkovo Institute of Science and Technology, Moscow, Russia

* Correspondence to karel.brinda@hms.harvard.edu

Abstract

Motivation

De Bruijn graphs play an essential role in computational biology, facilitating rapid alignment-free comparison of genomic datasets as well as reconstruction of underlying genomic sequences. Subsequently, an important question is how to efficiently represent, compress, and transmit de Bruijn graphs of the most common types of genomic data sets, such as sequencing reads, genomes, and pan-genomes.

Results

We introduce simplitigs, an effective representation of de Bruijn graphs for alignment-free applications. Simplitigs are a generalization of unitigs and correspond to spellings of vertex-disjoint paths in a de Bruijn graph. We present an easy-to-plug-in greedy heuristic for their computation and provide a reference implementation in a program called ProphAsm. We use ProphAsm to compare the scaling of simplitigs and unitigs on a range of genomic datasets. We demonstrate that simplitigs are superior to unitigs in terms of the cumulative sequence length as well as of the number of sequences, and that they are sufficiently close to the theoretical bounds for practical applications. Finally, we demonstrate that, when combined with standard full-text indexes, simplitigs provide a scalable solution for k-mer search in pan-genomes.

Availability

ProphAsm is written in C++ and is available under the MIT license from http://github.com/prophyle/prophasm.

Introduction

The advance of DNA sequencing started the golden age of biology in which phenomena previously unobservable can be studied on an unprecedented scale. However, sequencing capacity has been growing faster than computer performance and memory, and also faster than available human resources. Nowadays large amounts of sequencing data are available, but with a decreasing completeness and quality. In consequence, the traditional sequence-based representations and sequence alignment-based techniques [1,2] become less suitable for real-life scenarios due the space- and time-complexities they impose as well as due to their sequence-oriented nature in the world of datasets exhibiting graph structure.

An example is given by bacterial genomics. Modern large-scale studies of bacterial species comprise tens of thousands of sequenced isolates (see, e.g., [3], [4], or the Global Pneumococcal Project (pneumogen.net/gps/)). However, the information about isolates' genomes is almost always incomplete, as sequencing provides only partial observations of the genomes. While it is relatively straightforward to compute draft assemblies of bacterial genomes, the completion of the genomes is difficult. Due to repetitive regions, a full reconstruction from short reads is mathematically impossible even if sequencing errors were perfectly random. Long reads are often unavailable and reference sequences are of limited applicability due to the high variability and even unclear borders between species. While draft assemblies may be sufficient for many analyses, they are often not an ideal universal representation for a multitude of reasons. Most importantly, draft assemblies created using different assemblers are not directly comparable and this can introduce false differential signals into studies. In many scenarios it is therefore desirable to work with a graph representation obtained directly from a sequencing experiment without trying to assemble the genome.

De Bruijn graphs belong to the most popular graph representations of genomic datasets. They are defined as directed graphs $G = (V, E)$ where V is the set of all k-mers (i.e., subwords of a fixed length k) occurring in the dataset with edges connecting a vertex v to a vertex w if there is a $k - 1$ long prefix-suffix overlap between these v and w. As follows from the definition, we can associate a de Bruijn graph with the underlying k-mer set and edges can be defined implicitly (unlike the edge-centric definition where k-mer sets are associated with edges [5]). In this paper, we consider only vertex-centric graphs.

De Bruijn graphs feature remarkable properties. First, their computation from data is easy and deterministic. Algorithms for enumerating and counting k-mers have been extensively studied and many programs are available [6–9]. If the datasets contain sequencing errors, the computation may also involve graph cleaning. This aims at removing those k-mers that are the result of sequencing errors and are due to their supposed randomness expected to be rare. Second, if k is chosen appropriately, de Bruijn graphs can capture substantial information about the entire molecules under sequencing as these correspond to (some of the) walks in the graphs, provided that sequencing was sufficiently deep. Third, de Bruijn graphs can be handled easily, which simplifies software development as well as dataset analysis and interpretation. These properties have led to a large variety of applications of de Bruijn graphs.

De Bruijn graphs have been widely studied in the context of sequence assembly [10–12]. Here, their construction is typically the first step to the reconstruction of the genomes and transcriptomes under sequencing from retrieved sequencing reads. Many modern assemblers (e.g., Spades [13], ABySS [14], Velvet [15], Minia [16], and MEGAHIT [17]) follow the de-Bruijn graph paradigm.

Alignment-free sequence comparison [18] is another major application of de Bruijn graphs, following the idea that similar sequences share similar k-mers, and comparing de Bruijn graphs thus provides a good measure of sequence or dataset similarity. This involves the use of de Bruijn graphs for variant calling and genotyping [19–23], transcript

abundance estimation [24], metagenomic classification [25–28]. The latter also demonstrates another particularity of de Bruijn graphs – their high ability to approximate the graph structure of pan-genomes. Indeed, reference databases of bacterial strains are often highly incomplete and noisy; nevertheless, k-mer-based classifiers perform best in inferring the abundance profiles among all classifiers [29], which also suggests that de Bruijn graphs can be used as a proxy to variation graphs. Conversely, de Bruijn graphs with a large k can be used for indexing variation graphs [30,31].

The importance of de Bruijn graph leads us to an important problem: their space-efficient representation. While general de Bruijn graphs may impose large space requirements, it has been shown that those of real datasets can be highly compressible. Indeed, given the linearity of DNA and RNA molecules and the nature of sequencing, genomic datasets exhibit the spectrum-like property; i.e., there exist long strings of which most of the dataset's k-mers are substrings [5].

In this paper, we study the problem of representation of de Bruijn graphs for alignment-free data analysis. Building on previous works [32,33], we propose *simplitigs* as an effective representation of de Bruijn graphs. Simplitigs provide a textual representation of the graph, representing each k-mer exactly once and facilitating easy indexing with standard full-text indexes. Simplitigs use the observation that in practical applications such graphs typically contain long paths. In contrast to unitigs, which are paths that do not contain any branching nodes, simplitigs can contain branching nodes.

Finally, we present ProphAsm, a tool for computing simplitigs for a given dataset, such as reads, genomes, pan-genomes or metagenomes. ProphAsm proceeds by building the associated de Bruijn graph in memory, followed by a greedy enumeration of maximum vertex-disjoint paths. We use ProphAsm to demonstrate that simplitigs are superior to unitigs in terms of the cumulative sequence length and of the number of sequences, and

that they are sufficiently close to the theoretical bounds for practical applications. The employed heuristic can be easily integrated in any software producing de Bruijn graphs.

Results

Simplitigs as an efficient representation of de Bruijn graphs

We developed the concept of simplitigs to efficiently represent de Bruijn graphs for alignment-free applications (**Figure 1**). Simplitigs are a generalization of unitigs and corresponds to spellings of vertex-disjoint paths covering the de Bruijn graph; consequently, maximum simplitigs are such simplitigs that cannot be further compacted by merging (Methods). Note that unitigs and k-mers are also simplitigs, but not maximal, in general. The main conceptual difference between maximal simplitigs and maximal unitigs is that unitigs are limited by branching nodes (which are crucial for genome assembly), whereas simplitigs are not limited by this constraint. This allows for further compactification, with a benefit increasing proportionally to the amount of branching nodes in the graph.

We designed a greedy heuristic for the computation of simplitigs (**Algorithm 1,** Methods). At every step, it selects a k-mer from the current k-mer set and keeps extending it forward and then backward as long as possible while removing the already used k-mers from the set. This process is repeated until all k-mers are covered. We provide a reference implementation in a program called ProphAsm (github.com/prophyle/prophasm). The heuristic can be easily applied by any other software that outputs de Bruijn graphs or k-mer sets.

In the following sections, we use ProphAsm to compare maximal simplitigs with maximal unitigs on different types of data sets.

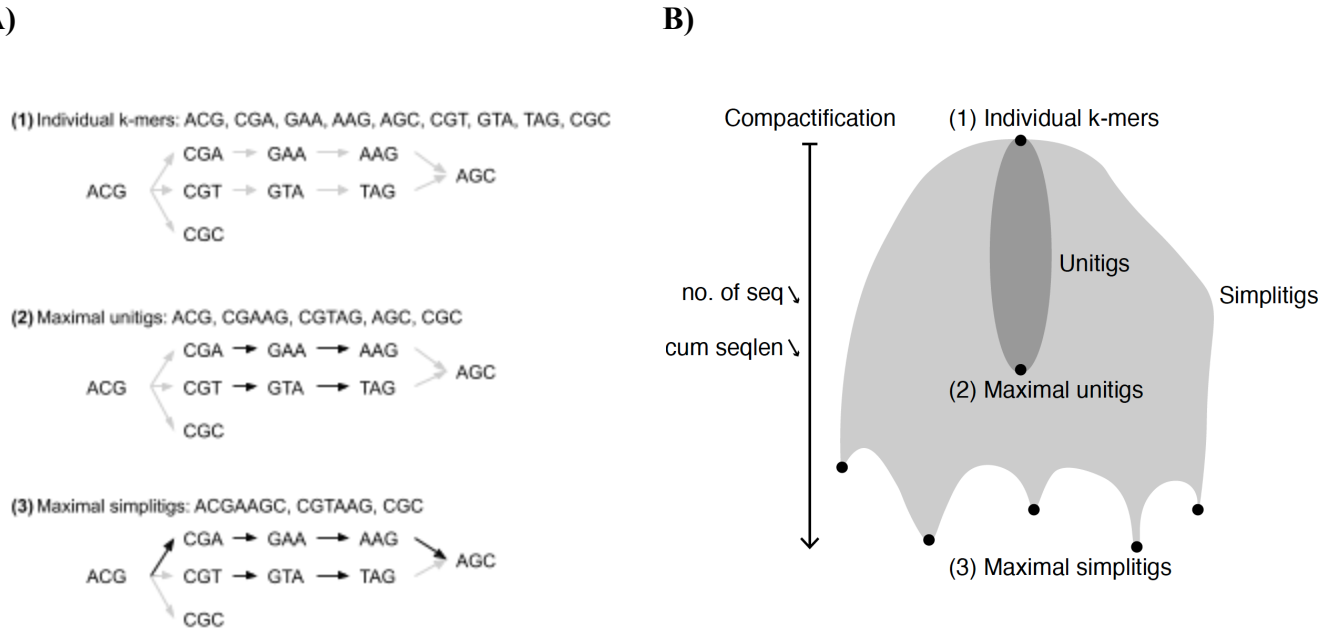

Figure 1. Simplitigs vs. unitigs and uncompacted k-mers. A) Simplitig subgraphs of de Bruijn graphs corresponding to individual kmers (1), maximal unitigs (2), and maximal simplitigs (3). Every component of a simplitig subgraph corresponds to a path and its spelling constitutes a simplitig (see Methods for more details). **B) Scheme of different types of simplitig subgraphs with respect to the degree of compactification of the k-mer set.** While unitigs (the dark grey area) correspond to compactification along non-branching nodes in the associated de Bruijn graph, simplitigs (the light and dark grey areas) can contain also branching nodes. When starting with individual k-mers, every step of compactification decreases the number of sequences by 1 and the cumulative length of sequences by $k-1$. Unlike maximal unitigs, maximal simplitigs are not determined uniquely and they may have even different cumulative lengths (corresponding to different local optima of compactification).

Algorithm 1. Greedy computation of maximal simplitigs for a k-mer set. In an iterative fashion, the algorithm draws an arbitrary k-mer from the set of k-mers K as a new simplitig, and then keeps extending it forwards and backwards as long as possible, while removing the already used k-mers from K.

```
Function extend_simplitig_forward (K, simplitig):
        extending = True
        while extending:
                extending = False
                q = simplitig[-k+1:]
                for x in ['A', 'C', 'G', 'T']:
                        kmer = q + x
                        if kmer in K:
                                extending = True
                                simplitig = simplitig + x
                                S.remove (kmer)
                                S.remove (reverse_complement (kmer))
                                break
        return S, s

Function get_maximal_simplitig (K, initial_kmer):
        simplitig = initial_kmer
        K.remove (kmer)
        K.remove (reverse_completement (kmer))
        K, simplitig = extend_simplitig_forward (K, simplitig)
        simplitig = reverse_completent (simplitig)
        K, simplitig = extend_simplitig_forward (K, simplitig)
        return K, simplitig

Function compute_simplitigs (kmers):
        K = {}
        for kmer in kmers:
                K.add (kmer)
                K.add (reverse_completement(kmer))
        simplitigs = {}
        while |K|>0:
                initial_kmer = K.random ()
                K, simplitig = get_maximal_simplitig (K, initial_kmer)
                simplitigs.add (simplitig)
        return simplitigs
```

Simplitigs of selected model organisms

We evaluated the simplitig representation on individual genomes of five model organisms for a range of k-mer lengths (**Figure 2,** Methods). The range was selected based on values that are most commonly used for alignment-free sequence comparison (see, e.g., [24,25,34]). For each organism and a k-mer length, we computed maximal simplitigs and unitigs, and compared them in terms of two basic characteristics: the number of sequences produced and their cumulative length. Whereas the former defines the number of records to be kept, the latter determines the total memory need. Note that both numbers are tightly connected (Methods, (eq 1)).

First, we analyzed the number of sequences produced (**Figure 2**, upper plots). We observe that for all datasets, as the k-mer size increases, the number of simplitigs grows and later decreases slowly while the number of unitigs grows rapidly at the beginning, and subsequently drops substantially, approaching the number of simplitigs. The maxima of functions corresponding to simplitigs and unitigs may (e.g., *C. elegans*) or may not (e.g., *B. mori*) be reached at the same k-mer size.

We then analyzed the cumulative sequence length (**Figure 2**, lower plots). The cumulative length is bounded from below by the number of k-mers in the genome plus $k-1$, corresponding to the theoretically maximum degree of compactification. In such a case, all k-mers would occur on the same simplitig; however, this is not attainable for most datasets. As we can observe and (eq 1) explains, the shapes of the curves in the lower plots copy the upper plots, while being only shifted up by a multiple of the theoretical lower bound function. When comparing the simplitig and unitig curves, we can observe the same patterns as for the number of sequences.

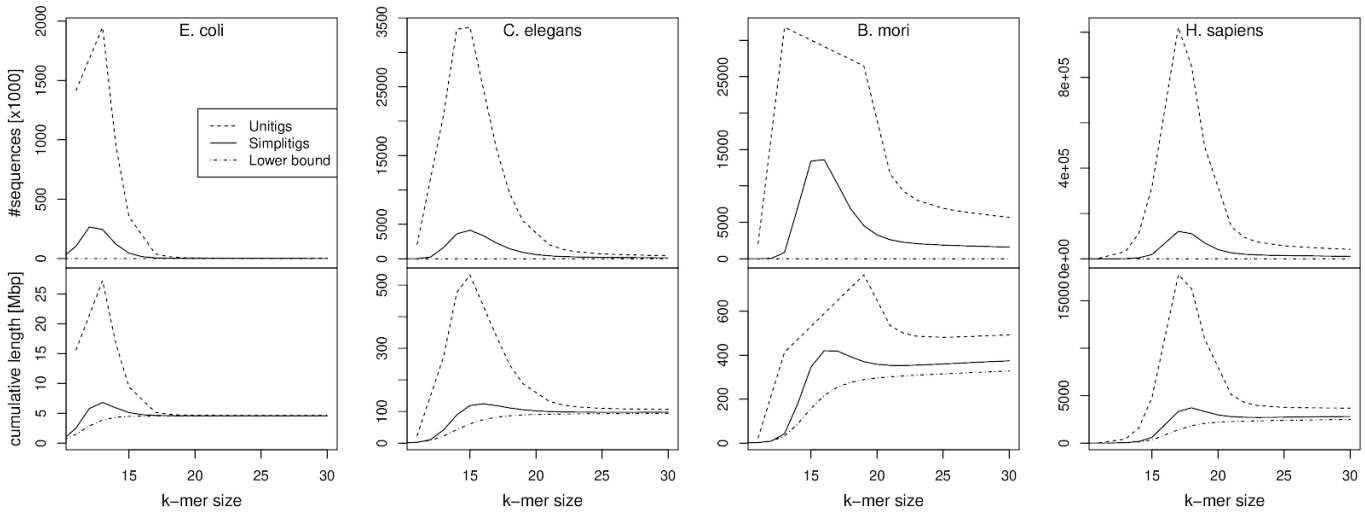

Figure 2. Comparison of the simplitig and unitig representations for selected model organisms and a range of k-mers. *Escherichia coli* (genome length: 4.64 Mbp), *Saccharomyces cerevisiae* (genome length: 12.2 Mbp), *Caenorhabditis elegans* (genome length: 100 Mbp), *Bombyx mori* (genome length: 482 Mbp), and *Homo sapiens* (genome length: 3.21 Gbp).

Overall, we observed that simplitigs always provide better performance than unitigs. In particular, they quickly approach the theoretical lower bounds for both characteristics tested. Every data set has a range of k-mer lengths where the difference between simplitigs and unitigs is striking, and after a certain threshold, the difference almost vanishes. While for short genomes this threshold is located for smaller k's (e.g., $k \approx 17$ for *E. coli*) than those typically used in alignment-free applications, in long genomes this threshold has not been attained on the tested range and seems to be substantially shifted towards large k-mers (e.g., *B. mori*). All this suggests that in practical applications, simplitigs are preferable for indexing individual genomes and the benefit is likely to increase with the genome size.

Simplitigs of bacterial pan-genomes

Computational pan-genomics has recently emerged as an important sub-branch of bioinformatics [35]. One of the motivations is the analysis of sequencing data in the context of whole species. Species are then represented using so-called pan-genome representations, i.e., reference structures including all within-species variation. De Bruijn graphs are particularly useful as pan-genomic references as they can be easily constructed from a variety of different data types, ranging from assembled reference sequences to the original sequencing reads reads. We sought to evaluate the usefulness of simplitigs for bacterial pan-genomes, which are particularly challenging due to their high diversity and variability.

We compared simplitig and unitig representations of the *Neisseria gonorrhoeae* pan-genome, as a function of the number of genomes included (**Figure 3**, Methods). We used 1,102 clinical isolates collected from 2000 to 2013 by the Centers for Disease Control and Prevention's Gonococcal Isolate Surveillance Project [36]; the data set comprises of draft assemblies from Illumina HiSeq reads. As expected, as the number of isolates and the associated variance grow, the number of sequences and their cumulative length grow as well, both for maximal unitigs and simplitigs. While simplitigs and unitigs perform comparably well when one bacterial genome is included (consistently with **Figure 2**), the improvement of unitigs over simplitigs grows and eventually stabilizes at approximately 1.5 as more genomes are included (**Figure 3**, bottom plot). On the other hand, the improvement in the number of sequences steadily decreases along the whole range and stabilizes at approximately 3.0.

To verify the generalizability of our findings, we repeated the experiment with the same dataset but for k=18 and also with 616 pneumococcal genomes from a carriage study of children in Massachusetts [37,38] with k=18 and k=31 (Methods). In all the cases, the results were qualitatively the same, although with small changes in the resulting relative improvements.

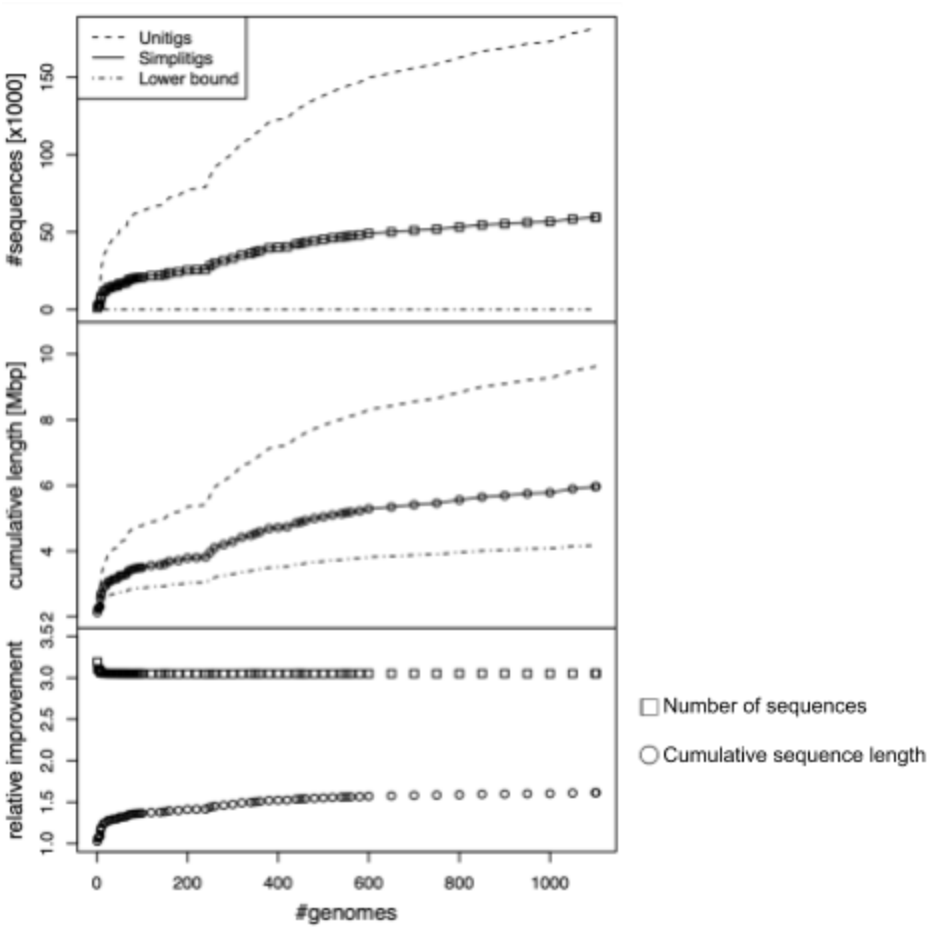

Figure 3. Pan-genomic scaling of maximal simplitigs and maximal unitigs for *Neisseria gonorrhoeae* and k=31. The first two plots show the number of sequences and their cumulative length as a function of the number of genomes, respectively. Lower bounds correspond to a hypothetical perfect case with a single simplitig containing all the k-mers. The third plot displays the relative improvement of simplitigs compared to unitigs.

Application of simplitigs for k-mer search in bacterial pan-genomes

Importantly, simplitig representation can be easily applied for k-mer lookup. As simplitigs and unitigs reduce k-mer lookup in a de Bruiijn graph to k-mer search in a set of sequences, any full-text search engine can be used. This includes several available highly optimized BWT indexes of read mappers [39–41]. BWT indexes [42] (i.e., indexes based on the Burrows-Wheeler Transform [43], sometimes also referred to as FM-indexes) are powerful compact representations of sequence data supporting fast string search. Although technical aspects of their implementation may vary (as well as their resulting performance), they are interchangeable in our applications. We used the index provided by BWA [40] to evaluate the performance of k-mer lookup in bacterial pan-genomes.

Single pan-genome

We first evaluated the performance of k-mer presence/absence queries on a single pan-genome (**Table 1,** Methods). We used the same *N. gonorrhoeae* draft genomes as previously to build a gonococcal k-mer pan-genome for five different k-mer sizes using three strategies: by merging the draft assemblies, by computing comprehensible unitigs, and by computing comprehensive simplitigs (**Table 1a**). For all of them, we constructed BWT indexes using BWA [40], queried ten million k-mers using BWA fastmap [44], and evaluated the resulting memory footprint and query performance (**Table 1b**).

Consistent with the previous experiments, simplitigs provided a clear improvement over unitigs (**Table 1a**). Maximal simplitigs improved 3.0x–4.9x the number of sequences and a 1.5x–2.1x the cumulative sequence lengths. Intuitively, we might assume the resulting memory footprint of BWA should be proportional to the cumulative sequence length, and therefore, the improvement in memory footprint was expected to be similar to the one of the cumulative sequence length. Surprisingly, the memory footprint improved substantially more with unitigs (2.7x – 5.6x) (**Table 1b**). To explain this phenomenon, it is important to understand that the underlying full-text engine has to keep information about individual sequences in memory as separate records and standard read mappers are

optimized for low numbers of references. As the number of reference sequences grows, it has a negative impact on both the memory footprint and query speed. However, since simplitigs provided 3.0x–4.9x improvement in the number of sequences over unitigs, it helped to alleviate this overhead. Overall, the comparatively high number of maximal unitigs observed throughout our experiments (**Figures 1 and 2**) provides a further argument for using unitigs as the preferable representation of k-mer sets.

Table 1. K-mer queries for the *N. gonorrhoeae* pangenome. a) Characteristics of the obtained unitigs and simplitigs. **b)** Time and memory footprint of BWA for k-mer queries (10M k-mers).

k	Draft assemblies		Unitigs		Simplitigs	
	seq count	cumulative length	seq count	cumulative length	seq count	cumulative length
15	79 k	2360 Mbp	440 k	9.25 Mbp	90.3 k	4.36 Mbp
19			190 k	6.86 Mbp	60.0 k	4.51 Mbp
23			180 k	7.65 Mbp	58.7 k	4.97 Mbp
27			180 k	8.61 Mbp	58.5 k	5.45 Mbp
31			182 k	9.63 Mbp	60.0 k	5.97 Mbp

b)

k	Draft assemblies		Unitigs		Simplitigs	
	time	mem	time	mem	time	mem
15	34 sec	3550 MB	42 sec	78 MB	24 sec	14 MB
19	50 sec		35 sec	37 MB	28 sec	12 MB
23	66 sec		41 sec	37 MB	32 sec	13 MB
27	81 sec		48 sec	38 MB	37 sec	14 MB
31	97 sec		56 sec	40 MB	42 sec	14 MB

Multiple pan-genomes

Finally, we evaluated the simplitigs performance for simultaneous indexing of multiple bacterial pan-genomes (**Table 2**, Methods). We downloaded all complete bacterial genomes from Genbank (as of December 2019; 10,502 genomes out of which we managed to download 9,570; Methods). We restricted ourselves to the complete genomes as the draft genomes in Genbank are known to be impacted by contamination [45]. We grouped individual genomes per species which resulted in 719 bacterial pan-genomes. We then computed simplitigs and unitigs for k=18 and k=31 for every species, merged the obtained representations, and calculated the same statistics as previously (**Table 2a**). Finally, we constructed BWT indexes using BWA, and measured the resulting k-mer lookup performance using the same ten million k-mers as previously (**Table 2b**).

In this case, the improvement of simplitigs over unitigs was 4.2x and 3.1x in the sequence count for k=18 and k=31, respectively, and 1.6x and 1.3x in the cumulative sequence length (**Table 2a**). For k=31 simplitigs provided 1.2x speedup and 1.8x improvement in memory consumption (**Table 2b**); for k=18, the speedup could not be evaluated (Methods). These results are consistent with the previous sections and provide further evidence that simplitigs are useful not only for storage, but also for fast k-mer lookup.

Table 2. K-mer queries for multiple pan-genomes indexed simultaneously. Bacterial pan-genomes were computed from the complete Genbank assemblies. **a)** Characteristics of the obtained unitigs and simplitigs. **b)** Time and memory footprint of BWA for k-mer queries (10M k-mers).

a)

k	Unitigs		Simplitigs	
	seq count	cumulative length	seq count	cumulative length
18	251 M	8.99 Gbp	59.0 M	5.71 Gbp
31	109 M	8.56 Gbp	35.5 M	6.37 Gbp

b)

k	Unitigs		Simplitigs	
	time	mem	time	mem
18	NA	21.0 GB	146	14.7 GB
31	179	23.4 GB	149	13.3 GB

Discussion

We introduced the concept of simplitigs, a generalization of unitigs, and demonstrated that simplitigs constitute a compact, efficient and scalable representation of de Bruijn graphs for commonly used genomic datasets. The two representations share many similarities. Both represent de Bruijn graphs in a lossless fashion, correspond to spelling of vertex-disjoint paths, and preserve k-mer sets. Being text-based and stored as FASTA files, both can be easily manipulated using standard Unix tools and indexed using full-text indexes. On the other hand, unlike unitigs, general simplitigs are not expected to have direct biological significance as neighboring segments of the same simplitig may correspond to distant parts of the same DNA molecule or even to different ones. Therefore, simplitigs are not applicable in many situations where unitigs are used, but on the other hand they show much better compression properties.

We provided ProphAsm, a tool implementing a greedy heuristic to compute maximal simplitigs. This heuristic is easy to implement in any software, which suggests its further use as a generic method for serialization of k-mer sets. The simplicity is in contrast to BCALM 2, the reference software for unitigs, where the complexity of the bi-directed de Bruijn graph model may complicate debugging; for instance, BCALM 2 does not support k-mer lengths that are divisible by four (as for December 2019, unsupported since 2017). As a downside, the naive implementation of the heuristic using a standard hashtable may run into memory issues. In our work, we have not encountered this, but memory consumption can be readily improved using more advanced data structures, similarly to what has been done for tools for unitig computation [33,46,47]. We note that ProphAsm is a spin-off of the ProPhyle software (https://prophyle.github.io/, [27]) for phylogeny-based metagenomic classification. Simplitig computation was already implemented in ProPhyle [48] for the purpose of efficient indexing of k-mers assigned to nodes of the phylogenetic tree.

The data presented in this paper are informative for understanding the scaling of computational resources as more sequencing data become available [49]. The studied gonococcal dataset constitutes a relatively complete snapshot of a bacterial population in a geographical region and at a given time scale. As such, it can be used to model the "state of completion" of k-mer pan-genomes. On the other hand, the multiple-pangenomes experiment provided insights about the resulting performance when a large number of pan-genomes is queried using a BWT index simultaneously. This allows to make predictions about the scaling for species where at present only a limited number of assemblies are available, but more data are likely to be generated in the future. Overall, with more data available, the comparative benefits of simplitigs over unitigs grow.

Besides the presented advantages, simplitigs also introduce several technical challenges related to the solution ambiguity (as illustrated in **Figure 1**). Whereas maximal unitigs are uniquely defined (up to the order and reverse complementing), this is not the case for maximal simplitigs. In the presented heuristic, the resulting maximal simplitigs and their characteristics depend on the order in which the initial k-mers are drawn from the underlying set. At every iteration, once a maximal simplitig is built, a new k-mer is drawn from the graph as the new initial k-mer. In the case of ProphAsm, this is an unordered set from the C++ standard library, which makes it difficult to implement reproducibly across platforms.

Modern bioinformatics applications of de Bruijn graphs often require multiple graphs considered simultaneously. The resulting structure is usually referred to as a colored de Bruijn graph [19] and its representations have been widely studied ([50–61]). Even though we touched this setting in the section Multiple pan-genomes, exploiting the similarity between individual de Bruijn graphs for further compression in simplitig-based approaches is to be addressed in future work.

With the growing interest in k-mer indexing of all genomic datasets [60], we anticipate the simplitig representation to be useful as a generic representation of de Bruijn graphs.

Methods

De Bruijn graphs

All strings are assumed to be over the alphabet $\{A, C, G, T\}$. A k-mer is a string of length k. For a string $s = s_1 ... s_n$, denote $s[i..j] = s_i ... s_j$. We also define $pref_k(s) = s_1 ... s_k$ and $suf_k(s) = s_{n-k+1} ... s_n$. For two strings s and t of length at least k, we define the binary connectivity relation $s \rightarrow_k t$ iff $pref_k(s) = suf_k(t)$.

Given a set K of k-mers, the *de Bruijn graph* of K is the directed graph $G = (V, E)$ with $V = K$ and $E = \{(u, v) \mid u \rightarrow_{k-1} v\}$. This definition of de Bruijn graphs is *node-centric*, as nodes are identified with k-mers and edges are implicit. Therefore, we can use the terms "k-mer set" and "de Bruijn graph" interchangeably.

Simplitigs

Consider a set K of k-mers and the corresponding de Bruijn graph $G = (V, E)$. A *simplitig graph* $G' = (V', E')$ is a subgraph of G that is acyclic and the in-degree and out-degree of any node is at most one. It follows from this definition that a simplitig graph is a vertex-disjoint union of paths called *simplitigs*. A simplitig is called *maximal* if it cannot be extended forward or backward without breaking the definition of simplitig graph. In more detail, a simplitig $u_1 \rightarrow_{k-1} u_2 \rightarrow_{k-1} ... \rightarrow_{k-1} u_n$ is maximal if the following conditions hold

- either u_1 has no incoming edges in G, or for any edge $(v, u_1) \in E$, v belongs to another simplitig,
- either u_n has no outgoing edges in G, or for any edge $(u_n, v) \in E$, v belongs to another simplitig.

A *unitig* is a simplitig $u_1 \rightarrow_{k-1} u_2 \rightarrow_{k-1} ... \rightarrow_{k-1} u_n$ such that each of the nodes $u_2, ..., u_n$ has in-degree 1 in graph G. A *maximal unitig* is defined similarly.

Greedy computation of simplitigs

The problem of computation of maximal simplitigs that would optimal in the cumulative sequence length corresponds to the maximum disjoint path problem which is known to be NP-hard [62]. Throughout this paper, a greedy approach was used for the simplitig computation (**Algorithm 1**). Simplitigs were constructed iteratively, starting from a random k-mer and being extended greedily forwards and backwards as long as possible. Note that **Algorithm 1** works in the bi-directed setting, in which canonical k-mers are used instead of standard k-mers. A formal definition of bi-directed de Bruijn graphs requires complex formalism (see, e.g., https://github.com/GATB/bcalm/tree/master/bidirected-graphs-in-bcalm2); however, the greedy heuristic works similarly in both setups and does not require this formalism. Therefore we resorted to the uni-directed model for the explanation of the concepts and to bi-directed model for the greedy heuristic.

Comparing simplitigs with unitigs

Simplitigs and unitigs were always compared in terms of the number of sequences produced and their cumulative length. Note that these numbers are related: assuming that the frequency of every k-mer is 1, then

$$cum_seq_len = \#kmers + (k-1)\,\#seqs \qquad \text{(eq 1)}$$

As the maximum disjoint path problem is known to be NP-hard [62], finding the optimal solutions can be highly intense computationally. However, we can provide the lower bound $\#kmers + k - 1$ corresponding to the maximum possible degree of compactification (i.e., a single simplitig covering all k-mers). In the situations where cumulative sequence length of unitigs approaches this bound, the greedy heuristic presented above is sufficient.

Correctness evaluation

The correctness of simplitigs can be verified using an arbitrary k-mer counter. Simplitigs are correct if and only if every k-mer is present exactly once and the number of distinct k-mers is the same as in the original datasets. To verify the correctness of ProphAsm outputs, we used JellyFish 2 [6].

Experimental evaluation – model organisms

Reference sequences for five selected model organisms were downloaded from RefSeq: *Escherichia coli* str. K-12 (accession: NC_000913.3, length: 4.64 Mbp), *Saccharomyces cerevisiae* (accession: NC_001133.9, length: 12.2 Mbp), *Caenorhabditis elegans* (accession: GCF_000002985.6, length: 100 Mbp), *Bombyx mori* (accession: GCF_000151625.1, length: 482 Mbp), and *Homo sapiens* (HG38, length: 3.21 Gbp). For each of them, simplitigs and unitigs were computed using ProphAsm and BCALM 2, respectively.

Experimental evaluation – pan-genomic scaling

First, 1102 draft assemblies of *N. gonorrhoeae* clinical isolates (collected from 2000 to 2013 by the Centers for Disease Control and Prevention's Gonococcal Isolate Surveillance Project [36], and sequenced using Illumina HiSeq) were downloaded from Zenodo (Grad 2019). Second, 616 draft assemblies of *S. pneumoniae* isolates (collected from 2001 to 2007 for a carriage study of children in Massachusetts, USA [37,38], and sequenced using Illumina HiSeq) were downloaded from the SRA FTP server using the accession codes provided in Table 1 in [38]. For each of these datasets, an increasing number of genomes was being taken, merged and simplitigs and unitigs computed using ProphAsm and BCALM 2, respectively, for k=18 and k=31. To avoid excessive resource usage the functions were evaluated at points in an increasing distance (for intervals [10, 100] and [100,+∞] only multiples of 5 and 20 were evaluated, respectively).

Experimental evaluation – fulltext k-mer queries

In the single pan-genome experiment, the same 1,102 assemblies of *N. gonorrhoeae* were merged into a single file. ProphAsm and BCALM 2 were then used to compute simplitigs and unitigs from this file for k=15, 19, 23, 27, 31. All three obtained FASTA files were used to construct a BWA index, which was then queried for k-mers using 'bwa fastmap -l {kmer-size}'. The k-mers were previously generated from the same pan-genome using DWGsim [63] (version 0.1.11, with the parameters '-z 0 -1 {kmer-size} -2 0 -N 10000000').

For the multiple pan-genome experiment, a list of available bacterial assemblies was downloaded from ftp://ftp.ncbi.nlm.nih.gov/genomes/genbank/bacteria/assembly_summary.txt. For all assemblies marked as complete, accessions were extracted and used for their download using RSync (files matching '*v?_genomic.fna.gz'). The obtained assemblies were then merged to a master file, which was then for computing simplitigs and unitigs using ProphAsm and BCALM 2. The obtained simplitig and unitig files were used to construct a BWA index and queried for the same k-mers as in the previous section using 'bwa fastmap -l {kmer-size}'. The times of loading the indexes into memory were measured separately and subtracted from the query times. With unitigs for k=18, bwa repeatedly crashed in the middle of k-mer matching from an unspecified reason.

Computational setup

All computation was performed on an iMac 4.2 GHz Quad-Core Intel Core i7 with 40 GB RAM. The reproducibility of computation was ensured using BioConda [64]. All benchmarking was performed using ProphAsm v0.1.0 and BCALM 2 v2.2.1. Times and memory footprint were measured using GNU time.

Implementation

ProphAsm is written in C++ and available under the MIT license from http://github.com/prophyle/prophasm.

Acknowledgements

This work was supported by the David and Lucile Packard Foundation.

References

1. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol. 1970;48: 443–453. doi:10.1016/0022-2836(70)90057-4

2. Smith TF, Waterman MS. Identification of common molecular subsequences. J Mol Biol. 1981;147: 195–197. doi:10.1016/0022-2836(81)90087-5

3. Petit RA, Read TD. Staphylococcus aureus viewed from the perspective of 40,000+ genomes. PeerJ. 2018;6: e5261. doi:10.7717/peerj.5261

4. Zhou Z, Alikhan N-F, Mohamed K, Fan Y, Achtman M, the Agama Study Group. The EnteroBase user's guide, with case studies on Salmonella transmissions, Yersinia pestis phylogeny, and Escherichia core genomic diversity. Genome Research. 2020. pp. 138–152. doi:10.1101/gr.251678.119

5. Chikhi R, Holub J, Medvedev P. Data structures to represent sets of k-long DNA sequences. 2019; 1–16. Available: http://arxiv.org/abs/1903.12312

6. Marçais G, Kingsford C. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics. 2011;27: 764–770. doi:10.1093/bioinformatics/btr011

7. Deorowicz S, Debudaj-Grabysz A, Grabowski S. Disk-based k-mer counting on a PC. BMC Bioinformatics. 2013;14: 160. doi:10.1186/1471-2105-14-160

8. Rizk G, Lavenier D, Chikhi R. DSK: k-mer counting with very low memory usage. Bioinformatics. 2013;29: 652–653. doi:10.1093/bioinformatics/btt020

9. Crusoe MR, Alameldin HF, Awad S, Boucher E, Caldwell A, Cartwright R, et al. The khmer software package: enabling efficient nucleotide sequence analysis. F1000Res. 2015; 1–12. doi:10.12688/f1000research.6924.1

10. Idury RM, Waterman MS. A New Algorithm for DNA Sequence Assembly. J Comput Biol. 1995;2: 291–306. doi:10.1089/cmb.1995.2.291

11. Pevzner PA. 1-Tuple DNA Sequencing: Computer Analysis. J Biomol Struct Dyn. 1989;7: 63–73. doi:10.1080/07391102.1989.10507752

12. Pevzner PA, Tang H, Waterman MS. An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences. 2001;98: 9748–9753. doi:10.1073/pnas.171285098

13. Bankevich A, Nurk S, Antipov D, Gurevich A a., Dvorkin M, Kulikov AS, et al. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. J Comput Biol. 2012;19: 455–477. doi:10.1089/cmb.2012.0021

14. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJM. ABySS: A parallel assembler for short read sequence data. 2009; 1117–1123. doi:10.1101/gr.089532.108.

15. Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome

Res. 2008;18: 821–829. doi:10.1101/gr.074492.107

16. Chikhi R, Rizk G. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. Algorithms Mol Biol. 2013;8: 22. doi:10.1186/1748-7188-8-22

17. Li D, Liu C-M, Luo R, Sadakane K, Lam T-W. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. Bioinformatics. 2015;31: 1674–1676. doi:10.1093/bioinformatics/btv033

18. Zielezinski A, Vinga S, Almeida J, Karlowski WM. Alignment-free sequence comparison: benefits, applications, and tools. Genome Biol. 2017;18: 186. doi:10.1186/s13059-017-1319-7

19. Iqbal Z, Caccamo M, Turner I, Flicek P, McVean G. De novo assembly and genotyping of variants using colored de Bruijn graphs. Nat Genet. 2012;44: 226–232. doi:10.1038/ng.1028

20. Bradley P, Gordon NC, Walker TM, Dunn L, Heys S, Huang B, et al. Rapid antibiotic-resistance predictions from genome sequence data for Staphylococcus aureus and Mycobacterium tuberculosis. Nat Commun. 2015;6: 10063. doi:10.1038/ncomms10063

21. Shajii AR, Yorukoglu D, William Yu Y, Berger B, Yu YW, Berger B. Fast genotyping of known SNPs through approximate k-mer matching. Bioinformatics. 2016;32: i538–i544. doi:10.1093/bioinformatics/btw460

22. Sun C, Medvedev P. Toward fast and accurate SNP genotyping from whole genome sequencing data for bedside diagnostics. Bioinformatics. 2019;35: 415–420. doi:10.1093/bioinformatics/bty641

23. Nordström KJV, Albani MC, James GV, Gutjahr C, Hartwig B, Turck F, et al. Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using k-mers. Nat Biotechnol. 2013;31: 325–330. doi:10.1038/nbt.2515

24. Bray NL, Pimentel H, Melsted P, Pachter L. Near-optimal probabilistic RNA-seq quantification. Nat Biotechnol. 2016;34: 525–527. doi:10.1038/nbt.3519

25. Wood DE, Salzberg SL. Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biol. 2014;15: R46. doi:10.1186/gb-2014-15-3-r46

26. Ames SK, Hysom DA, Gardner SN, Lloyd GS, Gokhale MB, Allen JE. Scalable metagenomic taxonomy classification using a reference genome database. Bioinformatics. 2013;29: 2253–2260. doi:10.1093/bioinformatics/btt389

27. Břinda K, Salikhov K, Pignotti S, Kucherov G. ProPhyle: An accurate, resource-frugal and deterministic DNA sequence classifier. Zenodo; 2017. doi:10.5281/zenodo.1045429

28. Corvelo A, Clarke WE, Robine N, Zody MC. taxMaps: comprehensive and highly accurate taxonomic classification of short-read data in reasonable time. Genome Res. 2018;28: 751–758. doi:10.1101/gr.225276.117

29. Ye SH, Siddle KJ, Park DJ, Sabeti PC. Benchmarking Metagenomics Tools for Taxonomic Classification. Cell. 2019;178: 779–794. doi:10.1016/j.cell.2019.07.010

30. Sirén J. Indexing Variation Graphs. 2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX). Philadelphia, PA: Society for Industrial and Applied Mathematics; 2017. pp. 13–27. doi:10.1137/1.9781611974768.2

31. Garrison E, Sirén J, Novak AM, Hickey G, Eizenga JM, Dawson ET, et al. Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nat Biotechnol. 2018;36: 875–881. doi:10.1038/nbt.4227

32. Chikhi R, Limasset A, Jackman S, Simpson JT, Medvedev P. On the Representation of De Bruijn Graphs. J Comput Biol. 2015;22: 336–352. doi:10.1089/cmb.2014.0160

33. Chikhi R, Limasset A, Medvedev P. Compacting de Bruijn graphs from sequencing data quickly and in low memory. Bioinformatics. 2016;32: i201–i208. doi:10.1093/bioinformatics/btw279

34. Břinda K, Callendrello A, Ma KC, MacFadden DR, Charalampous T, Lee RS, et al. Rapid inference of antibiotic resistance and susceptibility by genomic neighbour typing. Nature Microbiology. 2020. doi:10.1038/s41564-019-0656-6

35. Marschall T, Marz M, Abeel T, Dijkstra L, Dutilh BE, Ghaffaari A, et al. Computational pan-genomics: status, promises and challenges. Brief Bioinform. 2016; bbw089. doi:10.1093/bib/bbw089

36. Grad YH, Harris SR, Kirkcaldy RD, Green AG, Marks DS, Bentley SD, et al. Genomic Epidemiology of Gonococcal Resistance to Extended-Spectrum Cephalosporins, Macrolides, and Fluoroquinolones in the United States, 2000–2013. J Infect Dis. 2016;214: 1579–1587. doi:10.1093/infdis/jiw420

37. Croucher NJ, Finkelstein JA, Pelton SI, Mitchell PK, Lee GM, Parkhill J, et al. Population genomics of post-vaccine changes in pneumococcal epidemiology. Nat Genet. 2013;45: 656–663. doi:10.1038/ng.2625

38. Croucher NJ, Finkelstein JA, Pelton SI, Parkhill J, Bentley SD, Lipsitch M, et al. Population genomic datasets describing the post-vaccine evolutionary epidemiology of Streptococcus pneumoniae. Scientific data. 2015;2: 150058. doi:10.1038/sdata.2015.58

39. Langmead B, Trapnell C, Pop M, Salzberg SL. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. 2009;10: R25. doi:10.1186/gb-2009-10-3-r25

40. Li H, Durbin R. Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics. 2009;25: 1754–1760. doi:10.1093/bioinformatics/btp324

41. Li R, Yu C, Li Y, Lam T-W, Yiu S-M, Kristiansen K, et al. SOAP2: an improved ultrafast tool for short read alignment. Bioinformatics. 2009;25: 1966–1967. doi:10.1093/bioinformatics/btp336

42. Ferragina P, Manzini G. Opportunistic data structures with applications. Proceedings 41st Annual Symposium on Foundations of Computer Science. IEEE Comput. Soc; 2000. pp. 390–398. doi:10.1109/SFCS.2000.892127

43. Burrows M, Wheeler DJ. A Block-sorting Lossless Data Compression Algorithm. 1994.

44. Li H. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. Bioinformatics. 2012;28: 1838–1844. doi:10.1093/bioinformatics/bts280

45. Lu J, Salzberg SL. Removing contaminants from databases of draft genomes. PLoS Comput Biol. 2018;14: e1006277. doi:10.1371/journal.pcbi.1006277

46. Guo H, Fu Y, Gao Y, Li J, Wang Y, Liu B. deGSM: memory scalable construction of large scale de Bruijn Graph. IEEE/ACM Trans Comput Biol Bioinform. 2019; 1–1. doi:10.1109/TCBB.2019.2913932

47. Pan T, Nihalani R, Aluru S. Fast de Bruijn Graph Compaction in Distributed Memory Environments.

IEEE/ACM Trans Comput Biol Bioinform. 2018; 1–1. doi:10.1109/TCBB.2018.2858797

48. Břinda K. Novel computational techniques for mapping and classifying Next-Generation Sequencing data. PhD Thesis, Université Paris-Est. 2016.

49. Nasko DJ, Koren S, Phillippy AM, Treangen TJ. RefSeq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. Genome Biol. 2018;19: 165. doi:10.1186/s13059-018-1554-6

50. Bowe A, Onodera T, Sadakane K, Shibuya T. Succinct de Bruijn Graphs. 2012. pp. 225–235. doi:10.1007/978-3-642-33122-0_18

51. Holley G, Wittler R, Stoye J. Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage. Algorithms Mol Biol. 2016;11: 3. doi:10.1186/s13015-016-0066-8

52. Solomon B, Kingsford C. Fast search of thousands of short-read sequencing experiments. Nat Biotechnol. 2016;34: 300–302. doi:10.1038/nbt.3442

53. Muggli MD, Bowe A, Noyes NR, Morley PS, Belk KE, Raymond R, et al. Succinct colored de Bruijn graphs. Bioinformatics. 2017;33: 3181–3187. doi:10.1093/bioinformatics/btx067

54. Sun C, Harris RS, Chikhi R, Medvedev P. AllSome Sequence Bloom Trees. J Comput Biol. 2018;25: 467–479. doi:10.1089/cmb.2017.0258

55. Pandey P, Almodaresi F, Bender MA, Ferdman M, Johnson R, Patro R. Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index. Cell Syst. 2018;7: 201–207.e4. doi:10.1016/j.cels.2018.05.021

56. Yu Y, Liu J, Liu X, Zhang Y, Magner E, Lehnert E, et al. SeqOthello: querying RNA-seq experiments at scale. Genome Biol. 2018;19: 167. doi:10.1186/s13059-018-1535-9

57. Almodaresi F, Sarkar H, Srivastava A, Patro R. A space and time-efficient index for the compacted colored de Bruijn graph. Bioinformatics. 2018. pp. i169–i177. doi:10.1093/bioinformatics/bty292

58. Harris RS, Medvedev P. Improved representation of sequence Bloom trees. Bioinformatics. 2019. doi:10.1093/bioinformatics/btz662

59. Holley G, Melsted P. Bifrost – Highly parallel construction and indexing of colored and compacted de Bruijn graphs. bioRxiv. 2019; 1–19. doi:10.1101/695338

60. Bradley P, den Bakker HC, Rocha EPC, McVean G, Iqbal Z. Ultrafast search of all deposited bacterial and viral genomic data. Nat Biotechnol. 2019;37: 152–159. doi:10.1038/s41587-018-0010-1

61. Bingmann T, Bradley P, Gauger F, Iqbal Z. COBS: a Compact Bit-Sliced Signature Index. arXiv [cs.DB]. 2019. Available: http://arxiv.org/abs/1905.09624

62. Manuel P. Revisiting path-type covering and partitioning problems. arXiv [math.CO]. 2018. Available: http://arxiv.org/abs/1807.10613

63. Homer N. DWGSIM: Whole Genome Simulator for Next-Generation Sequencing. GitHub repository. 2010.

64. Grüning B, Dale R, Sjödin A, Chapman BA, Rowe J, Tomkins-Tinch CH, et al. Bioconda: sustainable and comprehensive software distribution for the life sciences. Nat Methods. 2018;15: 475–476.

564 doi:10.1038/s41592-018-0046-7

565