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 most common types of genomic data sets, such as sequencing reads, genomes, and pan-genomes.

Results

We introduce simplitigs, an efficient 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 implement it 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 are sufficiently close to 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.

Availability

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

Introduction

Advances in 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, of a decreasing completeness and quality though. In consequence, traditional sequence-based representations and sequence alignment-based techniques [1–3] have become less suitable for real-life scenarios due to the space- and time-complexities they impose as well as due to their sequence-oriented nature in the age 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., [4–6]). However, 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, completing the genomes is difficult. Due to repetitive regions, a full reconstruction from short reads is mathematically impossible even if the sequencing reads were error-free [7]. Long reads are often unavailable and reference sequences are of limited applicability due to the high variability of bacteria and 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 [8–10]. In many scenarios it is therefore desirable to move data analysis closer to the sequencing technology and work with graph representations obtained directly from raw reads without assembling the genomes.

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., substrings 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 v and w. As follows from the definition, a de Bruijn graph is defined by the underlying *k*-mer set and its edges can be defined implicitly (unlike the edge-centric definition where *k*-mer sets are associated with edges [11]). 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 [12–15]. 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, due to their supposed randomness, are 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 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 [16–18]. Here, their construction is typically the first step to the reconstruction of genomes and transcriptomes under sequencing from retrieved sequencing reads. Many modern assemblers (e.g., SPAdes [19], ABySS [20], Velvet [21], Minia [22], and MEGAHIT [23]) follow the de-Bruijn-graph paradigm.

Alignment-free sequence comparison [24] is another major application of de Bruijn graphs, following the idea that similar sequences share common *k*-mers, and comparing de Bruijn graphs thus provides a good measure of sequence or dataset similarity. This involves applications of de Bruijn graphs to variant calling and genotyping [25–29], transcript abundance estimation [30], and metagenomic classification [31–34]. The latter also demonstrates another particularity of de Bruijn graphs – their remarkable 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 among all classifiers in inferring abundance profiles [35], which also suggests that de Bruijn graphs can be used to represent pan-genomes. Furthermore, de Bruijn graphs with a large *k*-mer size can be used for indexing variation graphs [36,37].

The importance of de Bruijn graphs leads us to a key 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 *k*-mer datasets exhibit the so-called spectrum-like property: the existence of long strings of which most of the *k*-mers are substrings [11].

In this paper, we study the problem of representation of de Bruijn graphs for alignment-free data analysis. Building on previous works [38,39], we propose *simplitigs* as an effective representation of de Bruijn graphs. Simplitigs provide a “textual” representation of the graph, in the form of a set of sequences, 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 the 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 maximal vertex-disjoint paths. We use ProphAsm to demonstrate that simplitigs are superior to unitigs both in terms of the cumulative sequence length and the number of sequences, and that they are sufficiently close to theoretical bounds in practical applications. The employed heuristic can be easily integrated into 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 correspond to spellings of vertex-disjoint paths covering a given de Bruijn graph; consequently, maximal 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 an 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.

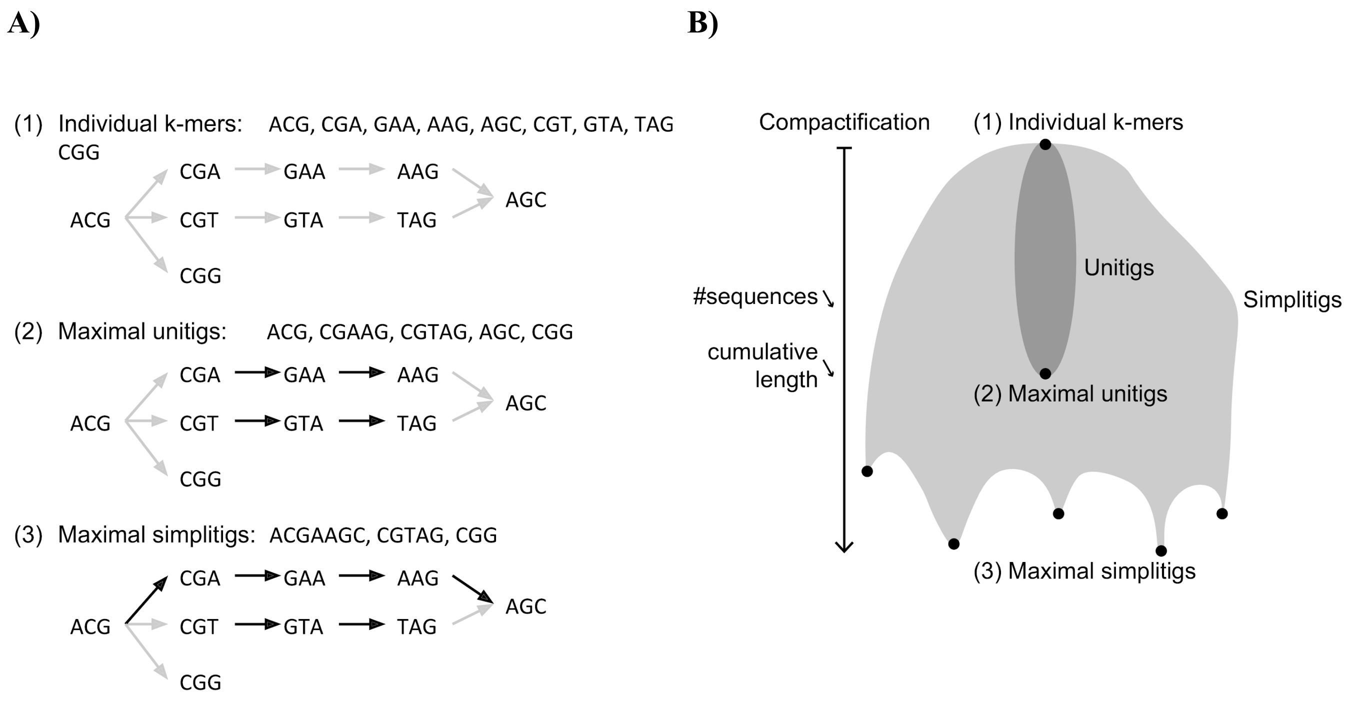
A)
(1) Individual k-mers: ACG, CGA, GAA, AAG, AGC, CGT, GTA, TAG CGG
CGA
GAA
AAG
AGC
ACG
CGT
GTA
TAG
CGG
(2) Maximal unitigs: ACG, CGAAG, CGTAG, AGC, CGG
(3) Maximal simplitigs: ACGAAGC, CGTAG, CGG
B)
Compactification
#sequences
cumulative length
(1) Individual k-mers
Unitigs
Simplitigs
(2) Maximal unitigs
(3) Maximal simplitigs

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 also contain 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 a *k*-mer from the set of canonical *k*-mers K, uses it as a new simplitig, and then keeps extending the simplitig forwards and backwards as long as possible, while removing the already used canonical *k*-mers from K.

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

Function get_maximal_simplitig (K, initial_kmer):
        simplitig = initial_kmer
        K.remove (initial_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 (canonical(kmer))
        simplitigs = {}
        while |K| > 0:
                initial_kmer = K.pop ()
                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 six model organisms for a range of *k*-mer lengths (**Figure 2,** Methods). Understanding the scaling based on the *k*-mer length is important for practical applications; the *k*-mer size is typically chosen with respect to the used sequencing technology and genomic diversity. The range for our experiments was selected based on values that are most commonly used for alignment-free sequence comparison (see, e.g., [30,31,40]). 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 needed. Note that the two 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 then decreases slowly. The number of unitigs grows rapidly at the beginning, and subsequently drops substantially, approaching the number of simplitigs. The cumulative length (**Figure 2**, lower plots) 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 factor of the theoretical lower bound. When comparing the simplitig and unitig curves, we can observe the same patterns as for the number of sequences.

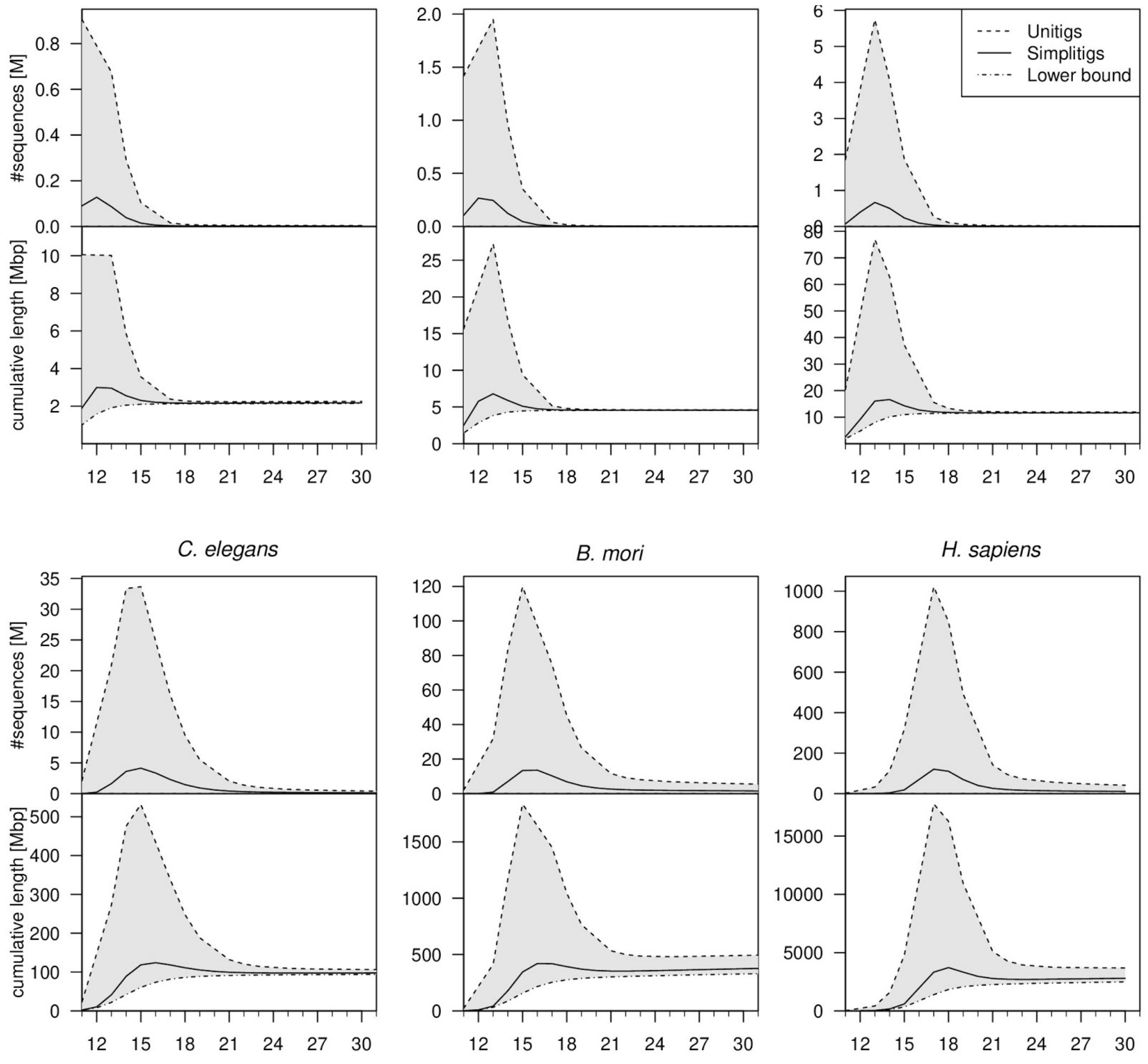

Figure 2. Comparison of the simplitig and unitig representations for selected model organisms and a range of *k*-mers. The number of sequences and their cumulative length for representation obtained by ProphAsm, BCALM 2 and the theoretical lower bound for six model organisms ordered by their genome size: *S. pneumoniae* (2,22Mbp), *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). The area highlighted in grey shows the discrepancy between the maximal unitigs and the theoretical lower bound.

Note that the maxima of both functions occur at (or are very close to) the value $k = log_4 G$, where G is the genome size. This is readily explained, as for values of k up to $log_4 G$, an overwhelming fraction of all 4^k *k*-mers belong to the genome, which makes the de Bruijn graph branch at nearly every node. As a consequence, unitigs are essentially reduced to individual *k*-mers, and their number grows exponentially. Starting from $k = log_4 G$, the number of *k*-mers is bounded by the genome length, and they begin to form longer non-branching paths in the graph, which drives down the number of unitigs. Importantly, however, the number of unitigs and their total size keep being much larger than those of simplitigs even for larger values of k, especially for large eukaryotic genomes.

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 at smaller *k*-mer lengths than those typically used in alignment-free applications (e.g., $k \approx 17$ for *E. coli*), for 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 [41]. 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. 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 for the *k*-mer length 31 (**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 [42]; the data set comprises 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 (consistent with **Figure 2**), the improvement of simplitigs over unitigs grows in the cumulative length as more genomes are included and eventually stabilizes at a factor of approximately 1.5 (**Figure 3**, bottom plot). On the other hand, the improvement in the number of sequences steadily decreases along the whole range and stabilizes at a factor of approximately 3.0.

To verify the generality of our findings, we repeated the experiment with the same dataset for the *k*-mer length 18 and also with 616 pneumococcal genomes from a carriage study of children in Massachusetts [43,44] with the *k*-mer lengths 18 and 31 (Methods). In all cases, the results were qualitatively the same, except for small changes in the resulting relative improvements.

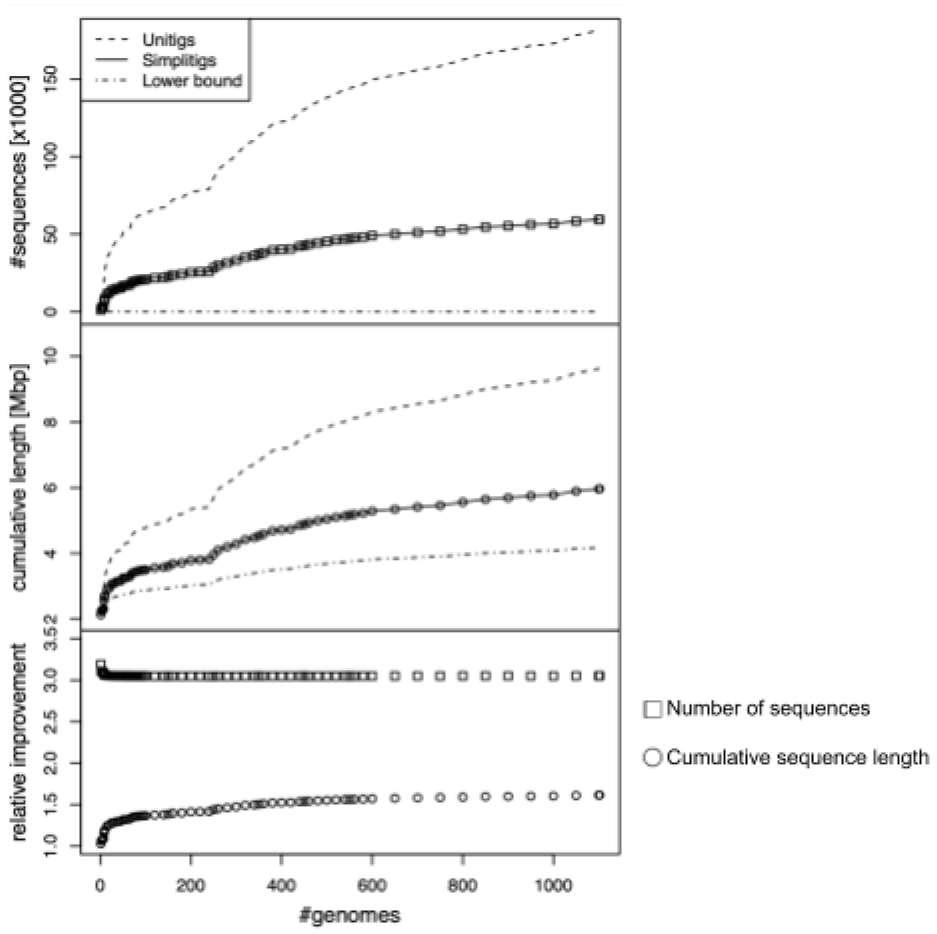

Figure 3. Pan-genomic scaling of maximal simplitigs and maximal unitigs for *Neisseria gonorrhoeae* and *k* = 31. The first two plots show the number of sequences and their cumulative length as a function of the number of genomes, respectively. Lower bounds correspond to a hypothetical perfect case with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

Any sequence data can be searched for *k*-mers using full-text indexes. Importantly, the simplitig representation can accelerate the *k*-mer lookup in datasets with redundant *k*-mer content by removing these redundancies, which we show on the example of *k*-mer look up in bacterial pan-genomes.

The most popular compact and powerful indexes supporting fast string search are BWT indexes [48], i.e., indexes based on the Burrows-Wheeler Transform [49], sometimes also referred to as FM-indexes. Many highly optimized implementations were developed for read mapping (e.g., [45–47]); in our experiments we used the BWA index [46], following the widespread use and superior performance.

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 genome assemblies 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 [46], queried ten million *k*-mers using BWA fastmap [50], 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 3.0×–4.9× the number of sequences and a 1.5×–2.1× the cumulative sequence lengths. Intuitively, 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 (2.7× – 5.6×) (**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 3.0×–4.9× 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 simplitigs as the preferable representation of *k*-mer sets.

Table 1. *K*-mer queries for the *N. gonorrhoeae* pan-genome. 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**	
	# sequences [$\times 10^3$]	**cumulative length** [Mbp]	**# sequences** [$\times 10^3$]	**cumulative length** [Mbp]	**# sequences** [$\times 10^3$]	**cumulative length** [Mbp]
15	79	2,400	440	9.3	90	4.4
19			190	6.9	60	4.5
23			180	7.7	59	5.0
27			180	8.6	59	5.5
31			180	9.6	60	6.0

b)

k	**Draft assemblies**		**Unitigs**		**Simplitigs**	
	time [sec]	**mem** [MB]	**time** [sec]	**mem** [MB]	**time** [sec]	**mem** [MB]
15	34	3,600	42	78	24	14
19	50		35	37	28	12
23	66		41	37	32	13
27	81		48	38	37	14
31	97		56	40	42	14

Multiple pan-genomes

Finally, we evaluated the performance of the simplitig representation 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 largely impacted by contamination [51–53]. We grouped individual genomes per species which resulted in 719 bacterial pan-genomes. We then computed simplitigs and unitigs for every species, merged the obtained representations, and calculated the same statistics as previously (**Table 2a**); we performed this experiment for the *k*-mer lengths 18 and 31. Finally, we constructed BWT indexes using BWA, and measured the resulting *k*-mer lookup performance using the same ten million *k*-mers as in the previous section (**Table 2b**).

In this case, the number of sequences was reduced by a factor of 4.2× and 3.1× and the cumulative sequence length by a factor of 1.6× and 1.3× for $k = 18$ and $k = 31$, respectively (**Table 2a**). For $k = 31$ simplitigs provided 1.2× speedup and 1.8× 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 (10 million *k*-mers).

a)

k	**Unitigs**		**Simplitigs**	
	# sequences [$\times 10^6$]	**cumulative length** [Gbp]	**# sequences** [$\times 10^6$]	**cumulative length** [Gbp]
18	250	9.0	59	5.7
31	110	8.6	36	6.4

b)

k	**Unitigs**		**Simplitigs**	
	time [s]	**mem** [GB]	**time** [s]	**mem** [GB]
18	NA	21	146	15
31	179	23	149	13

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. Not all situations allow unitigs to be replaced by simplitigs, but where applicable, simplitigs show much better compression properties.

We provided ProphAsm, a tool implementing a greedy heuristic to compute maximal simplitigs from a *k*-mer set. 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 the unitig model, 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 ProphAsm heuristic using a standard hashtable may run into memory issues. However, the memory consumption can be readily improved using more advanced data structures, similarly to what has been done for tools for unitig computation [39,54,55].

We note that ProphAsm is a spin-off of the ProPhyle software (https://prophyle.github.io/, [33]) for phylogeny-based metagenomic classification. Simplitig computation is an important component of ProPhyle [56], allowing efficient indexing of *k*-mers assigned to nodes of the phylogenetic tree. Independently of the present work, simplitigs were also recently studied in [57] under the name “spectrum-preserving strings”.

The data presented in this paper highlight the scaling of computational resources as more sequencing data become available [58]. The studied gonococcal dataset constitutes a relatively complete image 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 pan-genomes experiment provided insights about the resulting performance when a large number of pan-genomes is queried simultaneously using a BWT index. This allows us 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 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 [25] and its representations have been widely studied ([59–70]). Even though we touched upon 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 [69], we anticipate the simplitig representation to be valuable as a generic compact 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 $s = s_1 \dots s_n$, we define $pref_k(s) = s_1 \cdots s_k$ and $suf_k(s) = s_{n-k+1} \cdots s_n$. For two strings s and t of length at least k, we define the binary connectivity relation $s \rightarrow_k t$ if and only if $pref_k(s) = suf_k(t)$. 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 spanning 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 $u_1 \rightarrow_{k-1} u_2 \rightarrow_{k-1} \dots \rightarrow_{k-1} u_n$ 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 and it is not its last vertex,
- either u_n has no outgoing edges in G, or for any edge $(u_n,v) \in E$, v belongs to another simplitig and it is not its first vertex.

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

Greedy computation of simplitigs

The problem of computing maximal simplitigs that are optimal in the cumulative sequence length corresponds to the vertex-disjoint path cover problem, which is known to be NP-hard in the general case [71] but the complexity is unknown for de Bruijn graphs. Throughout this paper, a greedy approach was used for the computation of simplitigs (**Algorithm 1**). Simplitigs were constructed iteratively, starting from an arbitrary *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). Since the greedy heuristic works similarly in both setups and does not require the extended formalism, we resorted to the uni-directed model for the explanation of the concepts.

Comparing simplitigs with unitigs

We compare simplitigs and unitigs 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

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

Finding the optimal solutions can be highly expensive computationally. However, we can easily 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 simplitigs 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 [12].

Experimental evaluation – model organisms

Reference sequences for six selected model organisms were downloaded from RefSeq: *S. pneumoniae* str. ATCC 700669 (accession: NC_011900.1, length 2.22 Mbp), *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, http://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.fa.gz, length: 3.21 Gbp). For each of them, simplitigs and unitigs were computed using ProphAsm and BCALM 2, respectively, for the range of *k*-mer sizes [11,31]. As the BCALM 2 algorithm does not support *k*-mer sizes that are multiples of 4, the corresponding experiments had been excluded from the evaluation. When applied to HG38, both programs also experienced in a single case of an integer overflow error: BCALM 2 and ProphAsm failed with $k = 31$ and $k = 16$, respectively.

Experimental evaluation – pan-genomic scaling

First, 1,102 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 [42], and sequenced using Illumina HiSeq) were downloaded from Zenodo [72]. Second, 616 draft assemblies of *S. pneumoniae* isolates (collected from 2001 to 2007 for a carriage study of children in Massachusetts, USA [43,44], and sequenced using Illumina HiSeq) were downloaded from the SRA FTP server using the accession codes provided in Table 1 in [44]. 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. This experiment was performed 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 (assemblies, simplitigs, and unitigs) 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 [73] (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 assemblies were then merged and the obtained master file then used 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 for an unspecified reason.

Computational setup

The model organism experiment was performed on the HMS O2 research high-performance cluster on nodes with 120 GB RAM. All other experiments were performed on an iMac 4.2 GHz Quad-Core Intel Core i7 with 40 GB RAM and an SSD disk. The reproducibility of computation was ensured using BioConda [74]. All benchmarking was performed using ProphAsm v0.1.0 and BCALM 2 v2.2.1 (commit c8ac60252fa). Times and memory footprint were measured using GNU time.

Implementation and availability

ProphAsm is written in C++ and available under the MIT license from http://github.com/prophyle/prophasm. The software package is also available from BioConda [74].

Acknowledgements

The authors thank Jasmijn Baaijens for careful reading and valuable comments. This work was supported by the David and Lucile Packard Foundation. Portions of this research were conducted on the O2 high-performance compute clusters, supported by the Research Computing Groups at Harvard Medical School.

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. Gotoh O. An improved algorithm for matching biological sequences. J Mol Biol. 1982;162: 705–708. doi:10.1016/0022-2836(82)90398-9
4. Petit RA, Read TD. Staphylococcus aureus viewed from the perspective of 40,000+ genomes. PeerJ. 2018;6: e5261. doi:10.7717/peerj.5261
5. Gladstone RA, Lo SW, Lees JA, Croucher NJ, van Tonder AJ, Corander J, et al. International genomic definition of pneumococcal lineages, to contextualise disease, antibiotic resistance and vaccine impact. EBioMedicine. 2019;43: 338–346. doi:10.1016/j.ebiom.2019.04.021
6. 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
7. Treangen TJ, Salzberg SL. Repetitive DNA and next-generation sequencing: computational challenges and solutions. Nat Rev Genet. 2011;13: 36–46. doi:10.1038/nrg3117
8. Salzberg SL, Phillippy AM, Zimin A, Puiu D, Magoc T, Koren S, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. Genome Res. 2012;22: 557–567. doi:10.1101/gr.131383.111
9. Bradnam KR, Fass JN, Alexandrov A, Baranay P, Bechner M, Birol I, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. Gigascience. 2013;2: 10. doi:10.1186/2047-217X-2-10
10. Alhakami H, Mirebrahim H, Lonardi S. A comparative evaluation of genome assembly reconciliation tools. Genome Biol. 2017;18: 93. doi:10.1186/s13059-017-1213-3
11. 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
12. 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
13. 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
14. 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
15. 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

16. 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

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

18. 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

19. 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

20. 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.

21. 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

22. 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

23. 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

24. 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

25. 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

26. 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

27. 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

28. 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

29. 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

30. 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

31. 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

32. 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

33. 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

34. 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

35. 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

36. 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

37. 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

38. 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

39. 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

40. 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

41. 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

42. 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

43. 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

44. 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

45. 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

46. Li H, Durbin R. Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics.

2009;25: 1754–1760. doi:10.1093/bioinformatics/btp324

47. 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

48. 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

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

50. 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

51. Merchant S, Wood DE, Salzberg SL. Unexpected cross-species contamination in genome sequencing projects. PeerJ. 2014;2: e675. doi:10.7717/peerj.675

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

53. Steinegger M, Salzberg SL. Terminating contamination: large-scale search identifies more than 2,000,000 contaminated entries in GenBank. bioRxiv. 2020. p. 2020.01.26.920173. doi:10.1101/2020.01.26.920173

54. 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

55. 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

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

57. Rahman A, Medvedev P. Representation of k-mer sets using spectrum-preserving string sets. bioRxiv. 2020. p. 2020.01.07.896928. doi:10.1101/2020.01.07.896928

58. 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

59. 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

60. 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

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

62. 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

63. 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

64. 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

65. 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

66. 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

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

68. 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

69. 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

70. 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

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

72. Grad Y. Data for "Genomic Epidemiology of Gonococcal Resistance to Extended-Spectrum Cephalosporins, Macrolides, and Fluoroquinolones in the United States, 2000-2013." Zenodo; 2019. doi:10.5281/ZENODO.2618836

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

74. 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. doi:10.1038/s41592-018-0046-7