Efficient graph-color compression with neighborhood-informed Bloom filters

Ingo Schilken,[1] Harun Mustafa,[1,2] Gunnar Rätsch,[1,2,*]
Carsten Eickhoff,[1,*] and Andre Kahles [1,2,*]

[1]ETH Zurich, Department of Computer Science, Zurich, Switzerland
[2]University Hospital Zurich, Biomedical Informatics Research, Zurich, Switzerland.

Abstract

Motivation

Technological advancements in high throughput DNA sequencing have led to an exponential growth of sequencing data being produced and stored as a byproduct of biomedical research. Despite its public availability, a majority of this data remains inaccessible to the research community through a lack efficient data representation and indexing solutions. One of the available techniques to represent read data on a more abstract level is its transformation into an assembly graph. Although the sequence information is now accessible, any contextual annotation and metadata is lost.

Results

We present a new approach for a compressed representation of a graph coloring based on a set of Bloom filters. By dropping the requirement of a fully lossless compression and using the topological information of the underlying graph to decide on false positives, we can reduce the memory requirements for a given set of colors per edge by three orders of magnitude. As insertion and query on a Bloom filter are constant time operations, the complexity to compress and decompress an edge color is linear in the number of color bits. Representing individual colors as independent filters, our approach is fully dynamic and can be easily parallelized. These properties allow for an easy upscaling to the problem sizes common in the biomedical domain.

Availability

A prototype implementation of our method is available in Java.

Contact

`andre.kahles@inf.ethz.ch`, `carsten.eickhoff@inf.ethz.ch`, `Gunnar.Ratsch@ratschlab.org`

1 Introduction

The revolution of high throughput DNA sequencing has created an unprecedented need for efficient representations of large amounts of biological sequences. In the next 5 years alone, the global sequencing capacity is estimated to exceed one exabyte [21]. While a large fraction of this capacity will be used for clinical and human genome sequencing that are well suited for reference based compression methods, the remaining amount is still impressively large. This remainder not only includes sequencing efforts on model and non-model organisms but also community approaches such as whole metagenome sequencing (WMS).

The next logical steps of data integration for genome sequencing projects are assembly graphs that help to gather short sequence reads into genomic contigs and eventually draft genomes. While assembly of a single species genome is already a challenging task [6], assembling a set of genomes from one or many WMS samples is even more difficult. Although preprocessing methods such as taxonomic binning [10] help to reduce its complexity, the task remains a challenge. A commonly used strategy to generate sequence assemblies is based on de Bruijn graphs that collapse redundant sequence information into a node set of unique substrings of length k (k-mers) and transform the assembly problem into the problem of finding an Eulerian path in the graph [20]. Especially in a co-assembly setting, where a mixture of multiple source sequence sets is combined and information in addition to the sequences needs to be stored, colored de Bruijn graphs form a suitable data structure, as they allow to associate one or many colors with each node or edge [13]. A second use case is to use such graphs for the efficient representation and indexing of multiple genomes, a so called *pan-genome* store [18].

Owing to the large size and excessive memory footprints of such graphs, recent work has suggested compressed representations for de Bruijn graphs based on approximate member query data structures [7, 3] or generalizations of the Burrows-Wheeler transform to graphs [5]. The latter is often referred to as the BOSS representation, an acronym of the authors' initials. The recent work on compressed colored De Bruijn graphs has followed this trend. Currently, there exist two distinct paradigms. The first is to compress the complete colored graph in a single data structure and the second to handle two separate (compressed) representations of graph and coloring. In the first group fall approaches such as the Bloom Filter Trie [12] for pan-genome representations or *deBGR* [19], an encoding for a weighted de Bruijn graph. The second group contains approaches such as *VARI* [16], that uses succinct Raman-Raman-Rao or Elias-Fano compression on the annotation vector, and *Rainbowfish* [1], that additionally takes into account the distribution of the annotations in the graph to achieve more efficient annotation compression rates. The recently introduced *Metannot* [17] is a succinct data structure based on wavelet tries. It performs similar to the other approaches, but allows for efficient handling of dynamic settings where annotation or underlying graph structure are subject to change.

For many genomics applications, for instance, the encoding of a pan-genome index for read labeling, an exact reconstruction of the colors is not necessary and an approximate recovery with high accuracy would be sufficient. In our work, we present a probabilistic compression scheme for an arbitrarily sized set of colors given an arbitrary underlying graph. Based on Bloom filters [4], a data structure for efficient approximate membership query with a one-sided error, we encode colors as bit vectors and store them as a set of filters. We further reduce the necessary storage requirements of the individual filters by maintaining weak requirements on their respective false-positive rates, which is subsequently corrected for using neighborhood information in the graph.

2 Approach

We propose a color-compression that takes advantage of the underlying graph, but does not depend on a specific type of graph. For practical purposes we will demonstrate our method, however, under the assumption that the underlying graph is a de Bruijn graph and we operate in the setting of pan-genome or metagenome representation.

We implement our reference metagenome as a *colored de Bruijn graph* (cDBG), which consists of a de Bruijn graph constructed from a collection of input sequences and an associated *annotation*. We represent this annotation as a bit matrix, associating each edge in the graph to a subset of predefined annotation classes.

To index the cDBG in a space-efficient manner, we employ the BOSS representation [5] of the DBG encoded with rank- and select-supporting succinct vectors, while the columns of the annotation matrix are stored in independent Bloom filters. As an error correction step, we employ an additional Bloom filter to indicate nodes at which neighboring edges change their colors.

Finally, we test the utility and scalability of the structures with a series of data sets derived from viral, bacterial, and human genomes.

2.1 Preliminaries and notation

Let Σ be an alphabet of fixed size (in the case of genome graphs, $\Sigma = \{\$, A, C, G, T, N\}$). Given a string $s \in \Sigma^*$, we use $s[i : j]$ to denote the substring of s from 1-based position i up to and including position j.

Given a collection of input strings $\mathcal{S} = \{s_1, \ldots, s_n\} \subset \Sigma^*$, we define the *input sequence* S to be the concatenation of the s_i with the delimiter $\underbrace{\$ \cdots \$}_{k}$. We also define

$$\tilde{s_i} = \underbrace{\$ \cdots \$}_{k} s_i \underbrace{\$ \cdots \$}_{k} \tag{1}$$

Finally, given bit vectors $a, b \in \{0, 1\}^m$, we use the notation $a \mid b$ and $a \ \& \ b$ to denote the bitwise OR and AND operators, respectively.

2.2 de Bruijn graph representation

Definition 2.1 *A* colored de Bruijn graph *of order k (where $k > 1$ is an integer) of the string S together with n associated color bits is an ordered tuple of the form*

$$cDBG = (V, E, A), \tag{2}$$

where

$$\begin{aligned} V &= \{S[i : i + k - 1] : i \in \{1, \ldots, |S| - k + 1\}\}, \\ E &= \{(v_i, v_j) \in V : v_i[2 : k] = v_j[1 : k - 1]\}, \\ A &\in \{0, 1\}^{|E| \times n}. \end{aligned}$$

The edge set E may also be defined in terms of substrings of S,

$$E' = \{S[i : i + k] : i \in \{1, \ldots, |S| - k\}. \tag{3}$$

It is clear that $E \cong E'$ and we use the map

$$\Psi : S[i : i + k] \mapsto (S[i : i + k - 1], S[i + 1 : i + k]) \tag{4}$$

to interconvert. The elements of V are also known as k-mers and the elements of E $(k + 1)$-mers.

Let $\{e_1, \ldots, e_{|E|}\}$ denote the reverse-lexicographically sorted elements of E. We then define the annotation matrix A such that

$$A_{ij} = 1 \iff \exists \ell \in \{1, \ldots, |\tilde{s_j}|\}, \tilde{s_j}[\ell : \ell + k] = e_i. \tag{5}$$

We represent the tuple (V, E) using the BWT-based BOSS representation [5]. Further details are provided in Supplemental Section A.

2.3 Bloom filter-based compression

Since the columns of the annotation matrix A encode set inclusion, we use a Bloom filter to probabilisticly store these vectors [4].

Definition 2.2 *Let X be a finite space of objects. A* Bloom filter *is a tuple $BF = (B, \mathcal{H})$, where $B \in \{0,1\}^m$ and $\mathcal{H} = \{h_1, \ldots, h_d\}$ is a collection of hash functions mapping each input to an element of $\{0, \ldots, m-1\}$. Let $B[i] \in \{0,1\}^m$ denote a bit vector in which only the i^{th} bit is set to one. On this structure, the operation* insert*, the relation $\in$, and the operator $\cup$ are supported,*

$$\begin{aligned} \text{insert}(BF, x) &= (B[h_1(x)] \mid \cdots \mid B[h_d(x)], \mathcal{H}), \\ x \in BF &\Leftrightarrow \text{insert}(BF, x) = BF, \\ (B_1, \mathcal{H}) \cup (B_2, \mathcal{H}) &= (B_1 \mid B_2, \mathcal{H}) \end{aligned}$$

Let $\mathcal{X}$ be a random variable defined on the universe X and let $\tilde{X} = \{x_1, \ldots, x_s\}$ be a sample drawn from $\mathcal{X}$ which is inserted sequentially into a Bloom filter BF. Then the *false positive probability* (FPP) can be approximated [14] as

$$P\left(\mathcal{X} \in BF | \mathcal{X} \notin \tilde{X}\right) \approx \left(1 - e^{-\frac{ds}{m}}\right)^d. \tag{6}$$

2.4 Neighborhood-informed compression

To reduce the FPP of the filters, we propose to exploit the fact that annotations of neighboring nodes in the graph tend to be the same. Based on the assumption that the annotation is constant over a segment of length ℓ, we can compute the intersection of the annotations over all nodes of the segment and obtain an annotation with much lower FPP. Following the argument in [14], the FPP for an annotation of a segment of length ℓ can be approximated as $\left(1 - e^{-\frac{ds}{m}}\right)^{d\ell}$, since there are effectively $d\ell$ independent hash functions in use that each lead to a reduced FPP.

We need an additional data structure to store nodes at which the annotations of incoming and outgoing edges differ. For this we introduce an additional bit vector called the *continuity vector* that stores for each node whether the colors of the incoming edges match the outgoing ones.

Definition 2.3 *Given the reverse-lexicographical ordering of the nodes $\{v_1, \ldots, v_{|V|}\}$, the continuity vector $C \in \{0,1\}^{|V|}$ is defined such that*

$$C_i = 1 \iff \exists e_{in}, e_{out} \in E', \Psi(e_{in})_2 = v_i \wedge \Psi(e_{out})_1 = v_i \wedge A_{in} \neq A_{out} \tag{7}$$

Using this vector, given an edge $e_i = (v_i, v_i') \in E$, its *continuity path* $\mathcal{N}(i) \subseteq E$ is defined recursively as follows:

$$(v_i, v_i') \in \mathcal{N}(i) \iff C_i \vee C_{i'} \tag{8}$$

$$(v_j, v_\ell) \in \mathcal{N}(i) \iff \exists (v_j', v_\ell') \in \mathcal{N}(i), (v_\ell = v_j' \wedge C_j) \vee (v_j = v_\ell' \wedge C_\ell) \tag{9}$$

We then encode C using a Bloom filter.

2.5 Decompression

We encode A as a collection of Bloom filters $\mathcal{B} = \{BF_1, \ldots, BF_n\}$ defined on the input space E. The annotation of the edge $e_i \in E$ is then queried as follows:

$$\text{query}(e_i) = (1_{e_i \in BF_1}, \ldots, 1_{e_i \in BF_n}) \tag{10}$$

We can then use this to define the annotation function as

$$\text{annotation}(e_i) = \text{query}(e_i)\ \&\ \text{query}(e_1')\ \&\ \cdots\ \&\ \text{query}(e_\ell') \qquad e_j' \in \mathcal{N}(i) \tag{11}$$

2.6 Dynamic properties

A desirable property for genomics applications is the dynamic representation of a given data structure. That is, to add or remove new color classes or to adapt the coloring to changes in the underlying graph structure. Coming back to the conceptual representation of the coloring as a bit matrix, we would like to allow for dynamic behavior for both changes in the edges (rows) and color classes (columns).

With the proposed method it is possible to efficiently extend the graph with additional nodes. New edges are added to the color bit Bloom filters and new discontinuity nodes are immediately added to the continuation Bloom filter. Using this strategy, it is important that the final size of the number of edges with a specific color is estimated correctly, as this determines the optimal size of the filter chosen.

In addition to the dynamic behavior on edges, our approach also supports dynamic coloring. When the colored de Bruijn graph is extended with additional color labels, each color bit simply gets a new Bloom filter. This will have no effects on the accuracy of the remaining colors. Vice versa, removing a color bit is as simple as ignoring or discarding the corresponding Bloom filter. A second advantage of this strategy is, that each color bit can be compressed with an independent compression rate. Hence, we can easily prioritize certain annotations on the De Bruijn graph by increasing their corresponding Bloom filter size and therefore their decompression accuracy.

2.7 Data

We used several different data sets to evaluate the behavior of our Bloom filter color compression. These four data sets originate from viruses (*Virus1000*), bacteria (*BacteriaSelect* and *BacteriaAll*) and humans (*chr22+gnomAD*) and were chosen to test the method on different coloring distributions, coloring sizes and coloring densities. We constructed de Bruijn graphs of order $k = 63$ for each data set.

The virus and bacteria data sets were both generated from publicly available GenBank [8] complete genome data. The *Virus1000* data set consists of 1000 randomly selected complete virus

genomes, whose resulting graph consists of several disjoint, linear paths. The *BacteriaSelect* and *BacteriaAll* data sets consist of 45 and 136 different bacterial strains from the genus *Lactobacillus*, respectively, which leads to a linear topology in the graphs with many shorter paths disconnecting and reconnecting from a main backbone path. For the human *chr22+gnomAD* dataset, chromosome 22 from the *hg19* assembly of the human reference genome was used as the main reference backbone, together with exome variants from the gnomAD dataset [9]. This results in a graph that has a similar structure as the *Virus1000* graph, scaling the number of nodes three-fold and reducing the total number of colors.

A summary of the data sets is shown in Table 1. The data sets have been used in previous work [17] and further information about the data sets and the list of all virus and bacteria strains that were used can be found in their appendices.

2.8 Parameters

For each data set, the size of each color Bloom filter is computed as a constant factor ε (shared between all filters) of the number of edges in the graph annotated with that color. Appropriate values of ε, which resulted in color Bloom filter collections with average accuracies of 95% and 99%, were determined through binary search.

3 Evaluation and Applications

This section empirically evaluates the performance of the proposed color compression scheme. Our experiments are based on a range of datasets originating from viruses (Virus1000), bacteria (*BacteriaSelect* and *BacteriaAll*) and humans (*chr22+gnomAD*) compiled by Mustafa *et al.* [17]. Table 1 gives a comprehensive overview of these collections in terms of number of nodes, edges, color bits and unique colors.

Table 1: Virus, bacteria and human datasets used for evaluation. b = color bits per edge, U = unique edge colors (bit combinations). *Virus1000* is composed of 1000 viral strains. *BacteriaSelect* and *BacteriaAll* consist of 45 and 136 bacterial strains, respectively. *chr22+gnomAD* uses chromosome 22 from the *hg19* assembly of the human reference genome as the main reference backbone, together with the gnomAD dataset [9].

	nodes	**edges**	b	U
Virus1000	15,342,369	15,360,442	1000	10585
BacteriaSelect	18,669,398	18,713,013	45	546
BacteriaAll	71,164,435	71,358,127	136	5190
chr22+gnomAD	54,386,415	54,723,569	10	595

We evaluate space efficiency of the color compression scheme in terms of bits per edge across the entire de Bruijn graph. As discussed earlier, the method is parameterized by its desired accuracy. Table 2 shows results across all datasets for accuracy settings of 95% and 99%. Note that decompression is slightly decreased for higher accuracy, as more context is needed. Here we count the annotation as being correct, if all annotation bits (i.e., the color) are correct. The annotation accuracy per annotation is much higher.

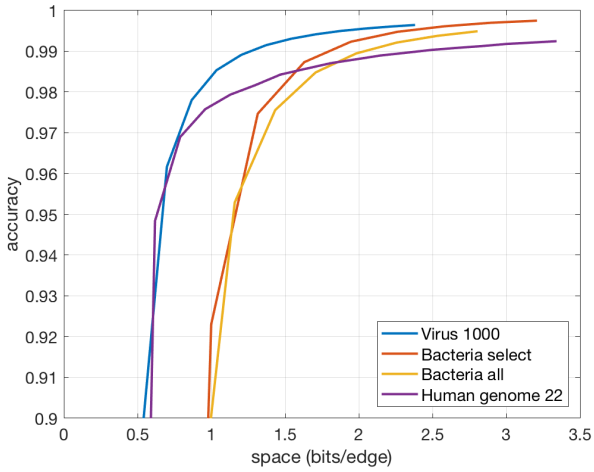
Table 2: Compression efficiency at color accuracy settings of 95% and 99.0%, respectively. The average individual Bloom filter false positive probability (FPP) is the ratio of set bits in the Bloom filter raised to the power of the number of hash functions (standard deviation smaller than 1% for all data sets). The average context length (including standard deviation) is the average number of queries to a color bit Bloom filter before a specific false positive bit is detected and removed from the color of a path.

	95% accuracy				
	bits/edge	**comp (s)**	**decomp (s)**	**avg. FPP**	**avg. context**
Virus1000	0.65	11	149	0.84	44 ± 7
BacteriaSelect	1.09	16	50	0.88	33 ± 9
BacteriaAll	1.16	77	280	0.82	28 ± 6
chr22+gnomAD	0.62	30	106	0.89	26 ± 11
	99% accuracy				
Virus1000	1.25	9	66	0.61	14 ± 2
BacteriaSelect	1.78	10	31	0.71	12 ± 3
BacteriaAll	1.98	61	165	0.63	11 ± 2
chr22+gnomAD	2.49	24	68	0.39	3.5 ± 1

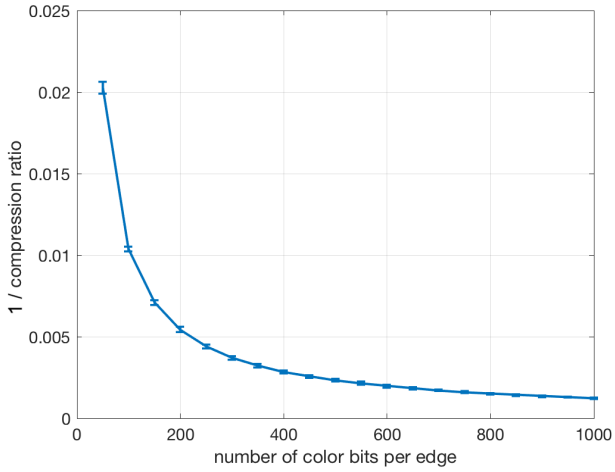
We can observe that even at high accuracy thresholds, compression to 1-3 bits per edge can be achieved. To measure the computational complexity of the proposed method, we report the single-thread compression and decompression times for the entire graph on a system with 36 cores of Intel(R) Xeon(R) CPU E5-2697 v4 (2.30GHz) that is part of ETH's shared high-performance compute systems. While decompression is generally the more costly of the two steps, both operations attain a throughput of several 100k edges per second, even on a single thread.

To further investigate the connection between target accuracy and compression ratio, Figure 1a plots accuracy as a function of the number of bits per edge invested into color Bloom filters. All datasets show a similar, steeply rising, behavior as we increase the relative filter sizes. This trend reaches its asymptote between 1 and 3 bits per edge at accuracy levels approaching 1.0. Similarly, Figure 1b focuses on the *Virus1000* collection and linearly increases the number of coloring bits per edge. For this data, we computed a chain of virus genome collections named *Virus50* to *Virus950* in steps of 50. *Virus50* consists of 50 randomly selected genomes from *Virus1000*, while subsequent collections are generated by randomly sampling additional sets of 50 genomes without replacement (i.e. *Virus100* is defined is a subsample of 100 genomes which contains *Virus50*). To report average compression ratios, 10 random draws of *Virus1000* were generated with different random seeds and used to compute the derived virus genome collections.

Finally, we close with a side-by-side comparison of the various de Bruijn graph color compression schemes presented in Section 1. In addition to these domain-specific methods, we include two popular general-purpose static compression methods, *gzip* and *bzip2*. Table 3 lists the number of bits per edge required to compress our four experimental collections. At an accuracy of 95% our method is considerably more space efficient, achieving compression ratios orders of magnitude greater than the competing methods. At 99% accuracy our approach performs comparably to Rainbowfish on the human genome collection while on all other collections we see a continued significant performance advantage of our method.



(a) Accuracy as a function of Bloom filter size in terms of bits per edge across all datasets.



(b) Compression ratio at 99% accuracy as a function of color bits per edge in *Virus1000*.

Figure 1: Compression accuracy vs. space efficiency.

Table 3: Baseline compression comparison. Compression rates are measured in average number of bits per edge.

	95%	**99.0%**	**VARI**	**RBFISH**	**WTr**	**gzip**	**bzip2**
Virus1000	0.65	1.25	N/A	36.011	22.756	38.239	12.606
BacteriaSelect	1.09	1.78	33.761	4.851	5.876	8.805	5.311
BacteriaAll	1.16	1.98	3.39	21.96	14.06	22.38	8.99
chr22+gnomAD	0.62	2.49	18.498	2.464	3.159	5.860	2.990

4 Conclusion

We have presented a probabilistic, compressed representation of a color encoding for arbitrary graphs, demonstrated on colored de Bruijn graphs. Our method uses approximate set representations for storing an arbitrary amount of annotations on the graph and leverages the graph topology and takes advantage of continuous colorings of neighboring nodes to improve the achieved compression ratios. Our representation can be efficiently decompressed and queried to retrieve the color of arbitrary paths in the graph. Although it is helpful to know the frequency of individual colors upfront to optimally choose the size of the individual Bloom filters used, this factor can be easily estimated from the size of the input data, allowing to directly build the full coloring.

We have shown the utility of our approach on different biological datasets, including data from virus, bacteria and human genomes, representing different classes of graph topologies and colorings. On all datasets we achieve comparable or strongly increased compression performance at very high level of decompression accuracy. Notably, our approach is fully dynamic and allows for an easy extension with additional labels / colors or for changes in the underlying graph structures, enabling the augmentation of large colored graphs with new annotations — a scenario commonly occurring in the genomics setting.

In future work we will adapt our method to better scale with dynamic changes. If a dataset grows rapidly in the number of edges, the decoding accuracy will eventually drop, eventually requiring a re-initialization of a larger Bloom filter. Currently this means to reload all elements into a larger filter. Further, despite being dynamic, our current representation does not allow for the removal of colors or edges from the graph. To support this we could replace the Bloom filters with other probabilistic set representations that allow for item removal [2, 11]. To further increase performance of the neighborhood-informed color compression, a separate continuity vector can be stored for each color Bloom filter (producing a *continuity matrix*) to further reduce the FPP. Given their greater sparsity compared to maintaing a single continuity vector, wavelet tries [17] may be an option for a dynamic structure to compress them losslessly. Lastly, an additional space improvement could be achieved with more space efficient probabilistic set representations such as compressed Bloom filters [15].

Acknowledgments

The authors would like to thank all members of the Biomedical Informatics group at ETH Zurich for valuable discussions, questions and feedback. Carsten Eickhoff is funded by the Swiss National Science Foundation Ambizione Program under grant agreement no. 174025. Harun Mustafa is funded by the Swiss National Science Foundation grant #407540_167331 "Scalable Genome Graph Data Structures for Metagenomics and Genome Annotation" as part of Swiss National Research Programme (NRP) 75 "Big Data". The authors declare no conflicts of interest.

References

[1] Fatemeh Almodaresi, Prashant Pandey, and Rob Patro. Rainbowfish: A succinct colored de bruijn graph representation. *bioRxiv*, 01 2017.

[2] Michael A Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P Spillane, and Erez Zadok. Don't thrash: how to cache your hash on flash. *Proceedings of the VLDB Endowment*, 5(11):1627–1637, 2012.

[3] Gaëtan Benoit, Claire Lemaitre, Dominique Lavenier, Erwan Drezen, Thibault Dayris, Raluca Uricaru, and Guillaume Rizk. Reference-free compression of high throughput sequencing data with a probabilistic de bruijn graph. *BMC bioinformatics*, 16(1):288, 2015.

[4] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[5] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. *Succinct de Bruijn Graphs*, pages 225–235. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[6] Keith R Bradnam, Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, Inanç Birol, Sébastien Boisvert, Jarrod A Chapman, Guillaume Chapuis, Rayan Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):10, 2013.

[7] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology : AMB*, 8:22–22, 2013.

[8] Karen Clark, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. Genbank. *Nucleic Acids Research*, 44(Database issue):D67–D72, 01 2016.

[9] Exome Aggregation Consortium, Monkol Lek, Konrad J Karczewski, and Eric V Minikel. Analysis of protein-coding genetic variation in 60,706 humans. *Nature*, 536(7616):285–291, 08 2016.

[10] Johannes Dröge and Alice C McHardy. Taxonomic binning of metagenome samples generated by next-generation sequencing technologies. *Briefings in bioinformatics*, 13(6):646–655, 2012.

[11] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88. ACM, 2014.

[12] Guillaume Holley, Roland Wittler, and Jens Stoye. Bloom filter trie: an alignment-free and reference-free data structure for pan-genome storage. *Algorithms for Molecular Biology : AMB*, 11:3, 2016.

[13] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 44(2):226–232, 2012.

[14] Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: building a better bloom filter. In *ESA*, volume 6, pages 456–467. Springer, 2006.

[15] Michael Mitzenmacher. Compressed bloom filters. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, PODC '01, pages 144–150, New York, NY, USA, 2001. ACM.

[16] Martin D. Muggli, Alexander Bowe, Noelle R. Noyes, Paul S. Morley, Keith E. Belk, Robert Raymond, Travis Gagie, Simon J. Puglisi, and Christina Boucher. Succinct colored de bruijn graphs. *Bioinformatics*, 33(20):3181–3187, 10 2017.

[17] Harun Mustafa, Andre Kahles, Mikhail Karasikov, and Gunnar Ratsch. Metannot: A succinct data structure for compression of colors in dynamic de bruijn graphs. *bioRxiv*, 2017.

[18] Gene Myers, Mihai Pop, Knut Reinert, and Tandy Warnow. Next generation sequencing (dagstuhl seminar 16351). In *Dagstuhl Reports*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[19] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. debgr: an efficient and near-exact representation of the weighted de bruijn graph. *Bioinformatics*, 33(14):i133–i141, 07 2017.

[20] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 98(17):9748–9753, 08 2001.

[21] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data: Astronomical or genomical? *PLoS Biol.*, 13(7):e1002195, July 2016.

A Supplemental Methods

Definition A.1 *The BOSS encoding of a DBG consists of the following four vectors:*

$$F = (e_1[k], \ldots, e_{|E|}[k]) \tag{12}$$

$$W = (e_1[k+1], \ldots, e_{|E|}[k+1]) \tag{13}$$

$$L = (1_{e_1[1:k] \neq e_2[1:k]}, \ldots, 1_{e_{|E|-1}[1:k] \neq e_{|E|}[1:k]}, 1) \tag{14}$$

$$B_i = 1 \iff \exists j < i, e_i[2:k+1] = e_j[2:k+1]. \tag{15}$$

For simplicity, the vector pairs (W, B) and (F, L) can be interleaved into $W^-, F^- \in (\Sigma \cup \Sigma-)^{|E|}$, respectively [1],

$$W_i^- = \begin{cases} W_i & \text{if } B_i = 0 \\ W_i- & \text{if } B_i = 1. \end{cases} \tag{16}$$

$$F_i^- = \begin{cases} F_i & \text{if } L_i = 1 \\ F_i- & \text{if } L_i = 0 \end{cases} \tag{17}$$

With this encoding,

Lemma A.1 *Forward graph traversal can be done using the first-last equivalence between F and W^- inherent to FM indices[5],*

$$e_i[2:k+1] = e_{\text{select}_{W^-}(F[i], \text{rank}_{F^-}(F[i], i))}[1:k] \qquad \forall i, L_i = 1. \tag{18}$$

[1] where $\Sigma- = \{c- : c \in \Sigma\}$