Advancements in practical k-mer sets: essentials for the curious

Camille Marchet

Advancements in practical *k*-mer sets: essentials for the curious

Camille Marchet[1]

DOI not yet assigned

Abstract

This paper provides a comprehensive survey of data structures for representing *k*-mer sets, which are fundamental in high-throughput sequencing analysis. It categorizes the methods into two main strategies: those using fingerprinting and hashing for compact storage, and those leveraging lexicographic properties for efficient representation. The paper reviews key operations supported by these structures, such as membership queries and dynamic updates, and highlights recent advancements in memory efficiency and query speed. A companion paper explores colored *k*-mer sets, which extend these concepts to integrate multiple datasets or genomes.

Keywords: *k*-mer, de Bruijn graph, index, hash table, filter, BWT, trie, MPHF, super-k-mer

[1]UMR9189 CRIStAL, Univ Lille, CNRS, Centrale, F-59000, Lille, France

Correspondence
camille.marchet@univ-lille.fr

1. Introduction

In the era of high-throughput sequencing, string algorithms are indispensable tools for the analysis of biological data. Sequencing technologies generate massive amounts of data by extracting numerous reads—short substrings of DNA or RNA from biological samples. These reads, which range in length from 50 to several thousand characters, are often subject to errors, making the analysis of sequencing data a complex problem. Central to this analysis is the task of string matching, where short, fixed-length substrings known as k-mers, are identified and analyzed. Over the past decade, k-mer-based methods have gained significant popularity due to their scalability and simplicity. These methods have been successfully applied across various biological domains, including genome [Bankevich et al., 2012] and transcriptome assembly [Bushmanova et al., 2019], transcript expression quantification [Patro et al., 2017a], metagenomic classification [Wood and Salzberg, 2014], and genotyping [Iqbal et al., 2012; Krannich et al., 2022]. Emerging applications include antibiotic resistance surveillance and detection [Bonin et al., 2023; Marini et al., 2022], and the curation k-mer signatures catalogs for cancer or other illnesses [Nguyen et al., 2021; Riquier et al., 2021].

A key challenge in k-mer-based methods is the efficient storage and querying of the vast sets of k-mers generated from sequencing data. As datasets grow in size and complexity, minimizing the storage requirements and query times for k-mer sets has become a crucial area of research. A previous work [Chikhi et al., 2019] provides a complete mathematical analysis of k-mer sets with space and time lower bounds. In this paper, I present a more broadly accessible survey of data structures designed for indexing k-mers, focusing on data-structures that have been used in practice. I categorize these structures and draw connections between them. I intentionally omit detailed discussions of their applications to specific biological problems, as this is the purpose of a companion article [Marchet, 2024]. Instead, I provide a high-level overview of the data structures themselves, offering insights into their underlying principles and practical implementations.

2. Preliminaries

2.1. De Bruijn graphs

In this paper, we consider a common definition of the de Bruijn graph in which distinct k-mers (words of size k) extracted from the sequences to be indexed are nodes (node-centric definition). More precisely, k-mers and their reverse complements are represented by the same node (the graph is called bi-directed, see Figure 1). This bi-directed definition is particularly convenient to build graphs from the general case of non-stranded sequencing data. The convention that the smallest k-mer (e.g. lexicographically) is considered the forward k-mer and the other is its reverse complement. Directed edges are drawn between nodes sharing k-1 exact overlaps on their forward or reverse representation. Other definitions of de Bruijn graph exists, with forward and reverse k-mer separated, or k-mers on edges [Rahman and Medvedev, 2022].

De Bruijn graphs are a scalable tool adapted for assembly since the advent of high-throughput short reads. They offer at most four possibilities (A, C, G, T) as the next nucleotide when assembling a contig, making graph construction and traversal feasible even for high coverages with respect to the size of genomes. But aside from assembly, de Bruijn graphs have general properties that are relevant for indexing datasets and genomes.

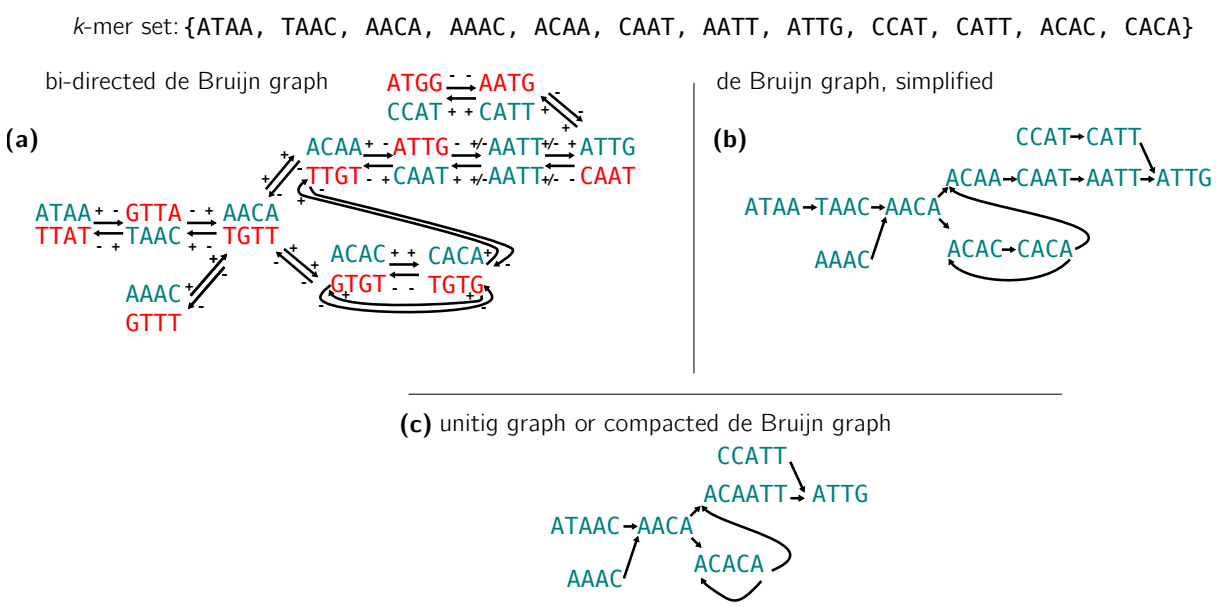

Figure 1 – Top of the figure: the k-mer set that is represented by the de Bruijn graphs. (a) A bi-directed de Bruijn graph. Forward k-mers are in blue, reverse in red. (+,+) egdes indicate an overlap between forward and forward k-mers, (-,-) are edges between reverse and reverse k-mers. (+,-) and (-,+) denote forward-reverse and reverse-forward nodes. I draw the reader's attention on the +/- symbols on some edges. This happens because they connect to and from a k-mer whose forward and reverse complement sequences are the same. This is one of the numerous edge case that must be taken care of when implementing de Bruijn graphs. (b) The simplified version that shows only one occurrence in the (forward, reverse) pairs, that is used in examples of this manuscripts. (c) The same graph but whose k-mers have been compacted in a unitig graph.

First, any set of distinct k-mers implicitly forms a de Bruijn graph since k-1 overlaps can be deduced from k-mers. De Bruijn graphs preserve the structure of the original sequence to some extent (hence facilitating genome and transcriptome assembly), which makes them highlight recognizable patterns of variants and enable error filtering through k-mer abundance and specific topological patterns. I recommend reading more k-mer key techniques reviewed in Jenike et al., 2024, with a focus on using k-mer set profiles for genome analysis.

Additionally, de Bruijn graphs allow the construction of shorter strings than simple concatenations of k-mers, which can be used to encode k-mer sets with fewer bits (reviewed in subsection 4.1). Perhaps the most well-known example is unitigs. A unitig is a stretch of DNA that is supposed to be unambiguous in the graph - meaning there are no branches in the path when connecting one k-mer to the next (see (c) in Figure 1). Compare for instance the number of symbols needed for the unitig `ACAATT` to the concatenation of its k-mers: `ACAA|CAAT|AATT`.

In the following, I review the different techniques practically utilised to represents sets of k-mers, or de Bruijn graphs for a single dataset. I spend much tome on k-mer tools whose main purpose is to count k-mers, such as KMC2/3 [Deorowicz et al., 2015; Kokot et al., 2017] or Jellyfish [Marcais and Kingsford, 2012]. They are quite often used as preliminary method for set representations reviewed here. I also omit some of the methods developed and embedded within genome assemblers and focus on standalone tools. Colored de Bruijn graphs, introduced in Iqbal et al., 2012, integrate multiple genomes or samples into a single structure, factorizing common parts and highlighting differences. Because they justify their own developments, I review them in a companion paper [Marchet, 2024].

3. Overview of *k*-mer sets

The plethora of methods for representing k-mer sets, each with slight variations, can be overwhelming for newcomers or researchers looking to utilize these methods in their analyses. Many tools incorporate other tools in their algorithms or implementations, creating a nested structure that complicates navigation. The goal of this section is to provide clarity by categorizing these methods to some extent, recognizing that such categorizations may not perfectly reflect reality but can help structure a coherent view (Figure 2).

3.1. Operations

Membership queries. A fundamental operation is checking the presence or absence of a given k-mer in the indexed collection. This is supported by all methods, with varying query times. Filter-based approaches can have false positives (thus called approximate membership queries), while other methods provide exact membership queries.
In terms of performance, the reviewed methods exhibit a range of time and space trade-offs.

Cache locality is often part of the discussion in modern methods, as it is crucial for optimizing program performance, especially when working with large datasets. It aims at maximizing the probability that for a given set of bits, the likely next accessed bits are close in the processor's cache. This minimizes delays and maximizes processing speed. Methods that have better cache locality often outperform others in practical scenarios, in particular when consecutive k-mers are queried (sometimes called batch queries, as opposed to single queries).
Navigation. Some structures also enable navigating the de Bruijn graph induced by the k-mer set, allowing to find adjacent k-mers and traverse the graph. This is particularly useful for assembly or variant detection applications. According to the structure design, navigational queries can be cheaper than membership queries.
Ranking. Some methods support ranking k-mers, for example to find the k-mer at a given position in the lexicographic ordering of the k-mer set.
Set operations. Operations like set union, intersection, and difference have been implemented in a few recent methods, enabling complex queries across multiple datasets.
Dynamic updates. While most methods are static, requiring a full rebuild upon changes in the k-mer set, a few dynamic structures allow efficient insertion and, more rarely, deletion of k-mers.
Count *k*-mers. Methods have been developped especially for this task, as this is an essential pre-processing to many pipelines. Counts can be approximate or exact.

3.2. Landscape of practical *k*-mer sets

We know the worst-case lower bound for representing a set of k-mers in a membership structure, under the assumption that k-mers are drawn from a uniform distribution, given by information theory [Conway and Bromage, 2011]: $\#bits = log_2\binom{4^k}{n}$, with n the number of k-mers in the set. Representing all canonical 31-mers from a human genome would require approximately 100 GB. We will see that this lower bound tells actually not much, as it is often beaten by specialized structures. Another bound has been set of structures that allow only navigational queries can go lower, down to 3.24 bits/k-mer [Chikhi et al., 2014].

Figure 2 (b), updated from Chikhi, 2021, presents the bit-per-k-mer cost for 21-mers.Key observations include that some methods make direct use of k-mer seen as lexicographic units, while others rely on k-mers as integers.

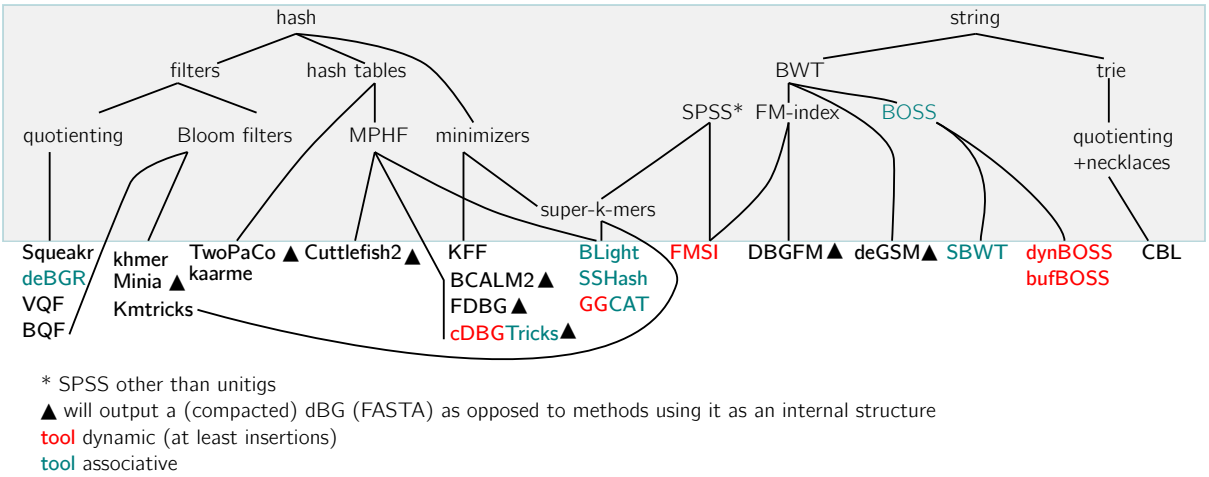

Figure 2 – Landscape of k-mer sets, starting from internal representation of k-mers based on strings or hashes. Tools to build (compacted) de Bruijn graphs (triangle) build the graph or unitigs from an input k-mer set and frequently output them in a FASTA format. K-mer sets allowing insertions (red), and sometimes deletions and set operations on an input k-mer set. Associative structures (blue) allow to pair k-mers with pieces of information, three of them are information specific: BQF, deBGR and Squeakr associate k-mers to count. The others are generalist k-mer dictionaries.

Strategies leveraging lexicographic context or redundancy of consecutive k-mers (reviewed in section 4) try to make the most of the data specificity: k-mers are extracted from a structured, non-random genomic context, sequencing data can contain a lot of redundancy and repeats.

Leverage string properties: example

For instance, consider the string `ACTGAGCTGAGCTGA`, which contains 7 5-mers. I chose it voluntarily redundant for the sake of the example. Applying a lexicographic transform on the string can change it into `AG$GGGGATTTAACCC` (a dummy `$` exists for technical reasons). `AG$GGGGATTTAACCC` has convenient runs of characters that can be further compressed, intuitively: `AG$(G,4)A(T,3)(A,2)(C,3)`.

Another way to reach a more compact representations uses the k-mer redundancy, as in the former unitig construction example: `ACAA|CAAT|AATT` $\rightarrow$ `ACAATT`.

In the second case, integer fingerprints can be stored instead of k-mers (reviewed in section 5). Fingerprints refer to compact, integer representations of k-mers. When a k-mer is added to the structure, its fingerprint (often a hash) is stored instead of the full k-mer. Lossy fingerprints (carrying less information than needed to retrieve the original k-mer) are one source of false positives, but represent a gain in bits per k-mer in the structure.

Fingerprints: example

Consider the k-mer `ATGGC`. The most straightforward, lossless, fingerprint is its binary encoding using the property that we can write `A` $\rightarrow$ `00`, `C` $\rightarrow$ `01`, `G` $\rightarrow$ `10`, `T` $\rightarrow$ `11`. It is therefore encoded on 2×5 bits as `0011101001`, or as the integer `489`. Consider the k-mer `AGGGC`, we obtain `0010101001` with an expected redundancy in the notation.

Fingerprints based on hashing have different properties and aim at 1-uniformity across the integer space and 2-determinism. `ATGGC` and `AGGGC` will likely be associated to very different values. We can also use lossy fingerprints to save space, for instance on 8 bits, and obtain for instance `ATGGC` $\rightarrow$ `00100001` and `AGGGC` $\rightarrow$ `01100000`.

Associativity (creating structures associating k-mer/value) can be achieved below the lower bound, costs fluctuate with input data structure, relying on the dataset's characteristics to optimize space. They rely on dictionary structure (key,value pairs structures with the k-mer as keys), or on structures that can associate k-mers using their ranks.

Another source of difference in the methods is that many methods are static, computed on a specific set and requiring a complete rebuild for any changes, a strategy achieving extremely low memory footprints. The possibility to add or remove k-mers without false positives is still rare, but available [Alanko et al., 2021; Alipanahi et al., 2021; Hannoush et al., 2024; Martayan et al., 2024; Sladkỳ et al., 2024], albeit at higher costs than the most recent structures. In a different approach, some methods allow updates at the price of a proportion of false positives when queried, which can be manageable with large datasets (reviewed in subsection 5.1.1).

Query performance is another critical aspect, that can benefit too from observations on the data specificity, notably by fitting the computer's cache size with k-mers that are likely to be queried altogether. The quickest methods, e.g. [Pibiri, 2022], query k-mers in a few hundreds of nanoseconds.

4. *K*-mers as strings

The methods reviewed in this subsection try and find some types of character redundancies in the k-mers in order to represent the k-mer space using less space.

4.1. Spectrum preserving string sets (SPSS)

For what matters in this article, *spectrum preserving string sets* have a confusing name. The goal of the techniques I review here is to encode a k-mer set within a string set. Indeed, methods constructing these sets usually do not preserve spectrum (k-mers with frequency), but only the k-mer set. I'd rather call them *set preserving string sets*[1] if I had a choice.

De Bruijn graphs allow constructing strings shorter than concatenated k-mers, enabling more compact k-mer set encoding. These string sets, have been formalized for k-mers by Rahman and Medevedev, 2021, although they appeared earlier in Břinda's work [Brinda, 2016]. Given a k-mer input, a spectrum preserving string set is a plain text representation from which all initial k-mers and nothing else can be spelled (Figure 3).

[1] as coined by A.L.

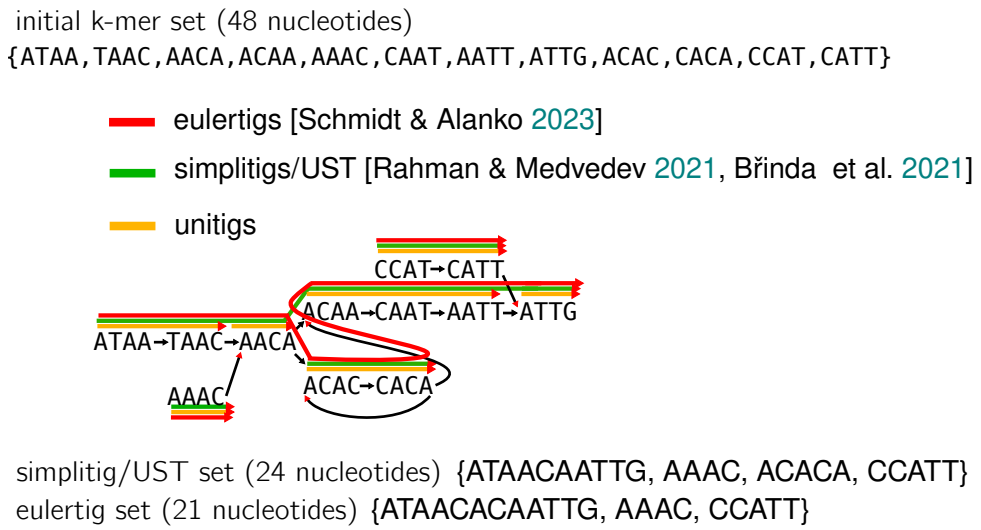

Figure 3 – Different examples of SPSS built from a same set of k-mers.

Paths in the de Bruijn graph can represent longer strings. Finding an SPSS involves identifying paths covering all graph nodes and processing nodes within each path with a compaction algorithm.

Compaction: example

Intuitively, compaction is merging consecutive strings as follows: if one k-mer ends spells "ATG" and the next k-mer is "TGA", they can be merged into a longer sequence composed of the first k-mer and the last base of the second, "ATGA", because they overlap.

An optimal solution for the SPSS problem does compactions so it minimizes the number of strings while ensuring each k-mer appears only once in the string set.

Unitigs, the product of spelling paths with no branch in the de Bruijn graph, are non-optimal SPSS, often used for more compact de Bruijn graph representations (unitig graphs are called compacted de Bruijn graphs), with different algorithms and trade-offs, including BCALM2 [Chikhi et al., 2016], TwoPaCo [Minkin et al., 2016], Cuttlefish2 [Khan et al., 2022], and GGCAT [Cracco and Tomescu, 2023]. Unitigs remain widespread because they are balanced options between shortest sequences and biological meaningfulness, they also are a intermediary component of short read assemblers. Moreover, they are often constructed after a k-mer filtering and error correction pass on the initial graph, and provide a cleaner set.

Recent heuristics, such as simplitigs [Břinda et al., 2021] and USTs [Rahman and Medevedev, 2021], come close to optimal solutions for the SPSS problem. They allow to store 31-mers from a human genome in 2 GB or less. Eulertigs [Schmidt and Alanko, 2023] further provide a linear algorithm for optimal solutions based on Eulerian cycles.

The most recent contribution, masked superstrings [Sladký et al., 2023], unify different proposals into a common framework. It generalizes the compactions to overlaps smaller than k-1, which creates spurious k-mer but increases the capacity to elongate strings. They use masks (binary arrays) to indicate which parts of the superstring correspond to the k-mers of interest. The GGCAT method for constructing de Bruijn graphs also proposes different options for SPSS. Before that, a different route was taken by matchtigs, that allow to re-use a k-mer several times to achieve larger superstrings [Schmidt et al., 2021].

While SPSS can represent k-mers compactly, they are not directly easy to query. They are combined with other techniques to create k-mer structures, namely, in the FMSI [Sladkỳ et al., 2024] index, and the KFF tool [Dufresne et al., 2022].

Example use cases of SPSS. SPSS are not usually used per se in pipelines, but rather as a piece of larger algorithms. For instance, recently, SPSS (USTs) were used within a compression algorithm that reduces k-mers and associated counts footprint [Rossignolo and Comin, 2024]. As we will see later in the article (subsection 5.2), they are a key component for modern k-mer hash tables.

4.2. BWT-based methods

The Burrows-Wheeler Transform (BWT) is a data transformation technique used primarily in data compression. It makes use of recurrent neighborhoods surrounding characters in a string to rearrange the string into runs of similar characters. This makes the string more amenable to compression. It is then possible to access any origin substring by querying the transform with a small quantity of auxiliary information in comparison to the original string size. An alternative strategy for k-mer sets is therefore using methods based on the BWT to index k-mer sets. These methods operate directly on sequences, grouping similar segments for compression. Among first proposals, many relied on an index based on the BWT, the FM-index [Ferragina and Manzini, 2005]. However, the BWT excels on large texts, not on sets of words, and its performances decline when there are errors in the dataset. That is why early uses of BWT for indexing k-mer sets have employed longer strings simplifying the k-mer sets, such as unitigs [Chikhi et al., 2014].

The BOSS structure [Bowe et al., 2012] specialized on k-mer sets and introduced a representation for edge-centric de Bruijn graphs (a definition where k-mers are on edges). The BOSS structure was also turned into a dynamic structure allowing insertions and deletions in k-mer sets, through a system of buffers and sporadic rebuild of the structure [Alanko et al., 2021; Alipanahi et al., 2021]. Despite early limitations in query performance, BOSS was superseded by a recent advancement, the SBWT [Alanko et al., 2023b] (Figure 4), offering node-centric representations, and enhancing query efficiency.

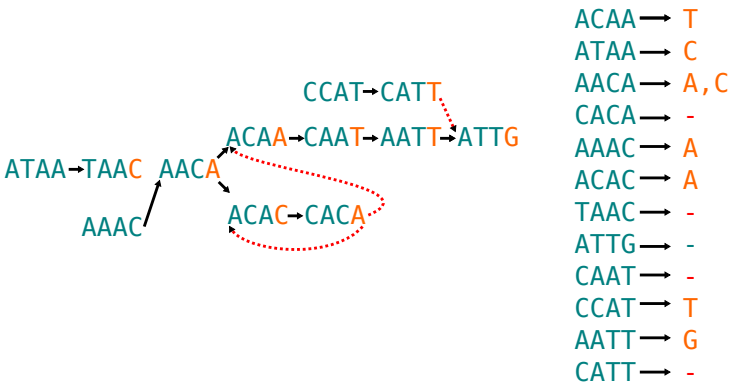

Figure 4 – Intuition of the SBWT of a de Bruijn graph. The SBWT retains connections between k-mers, and leaves some edges (dashed in red) when the k-mer already has another recorded parent. Last nucleotide of children k-mers for retained edges are associated to k-mers sorted lexicographically. The rightmost colored vector is in essence what is stored in the SBWT and allows retrieve k-mers. In practice, some more dummy nodes are needed and specific data structures are needed to retrieve efficiently the information.

The SBWT can index distinct 31-mers of the human genome in approximately 2 GB, and achieved competitive query speed when equipped with auxiliary query structures [Alanko et al., 2023a], for the price of a larger index. The BOSS and SBWT can be associative structures by using the colexicographic rank of k-mers as an index.

Example use cases of BWT-based methods. Burrows-Wheeler Transform based indexes have become fundamental tools in bioinformatics due to their ability to efficiently compress and search genomic data. BWT is used in countless applications, notably in read alignment tools. In the context of k-mer indexing, the BOSS structure was implied in different works dealing with surveillance of foodborne pathogens. It was integrated in a reference-free metagenomics SNP caller [Alipanahi et al., 2020], whose advantage is to be able to extract complex embedded SNPs, as in different samples of a beef production system for detecting antibiotic resistance genes. Similarly, an index based on the SBWT was embedded a in pipeline for genomic epidemiology dealing with mixed samples of a target pathogen [Mäklin et al., 2021].

4.3. Tries and variations

A trie is a tree-like data structure used to store a dynamic set of strings, where each node represents a single character of a string. Tries store strings in a way that allows common prefixes to be shared among different strings. Each path from the root to a leaf node in the trie represents a complete string. Tries are not too common for k-mer indexing, but appear from time to time in the literature [Agret et al., 2022; Holley et al., 2016].

Martayan et al., 2024 noticed that the necklace text transform [Sawada and Williams, 2017] increases the number of shared prefixes in a k-mer set. The transform is related to the BWT as it is also based on lexicographic rotations of words. Instead of a regular trie, the CBL structure is divided in two: prefixes and suffixes. Its stores once common prefixes to save space, and the rank of prefixes associates them with suffixes. The necklace transform also tends to improve cache efficiency, as consecutive k-mers will share prefixes.

The strategy where k-mers are split in a left (prefix) and right (suffix) part is also called quotienting, and more frequently used in hash-based techniques (see subsection 5.2).

CBL uses 200 GB to represent 31-mers of the human genome, trading space for full dynamicity. It is one of a few, with FMSI, that permits set operations on the k-mers of the structure: intersection, union, difference.

Example use cases of tries. CBL is used in the query builder Grimr [Ingels et al., 2024] that helps querying multiple samples based on meta-data constrained. Another trie structure was used to detect gene signature in a rice pangenome [Agret et al., 2022].

5. *K*-mers as hashes

The simplest k-mer integer representation is the succinct indicator bitmap used in Conway and Bromage, 2011. It uses the integer representation of k-mers to address them in a bit array, and can be compressed in a space close to the information-theoretic minimum while still allowing efficient access.

Other methods apply hash functions to k-mers first. These methods aim at populating tables with k-mers but have to deal with an initial skewed distribution, due to the repetitive and non-uniform content of k-mer sets with respect to the whole 4^k universe. Using directly the integer representation of the k-mers from these skewed distribution could lead to cluttered regions in tables, with impacts on space, insertion time and query efficiency. The reviewed methods rely on hashing, transforming k-mers into integers via hash functions, to uniformly fill the allocated table or bit set.

5.1. Structures with false positives: filters

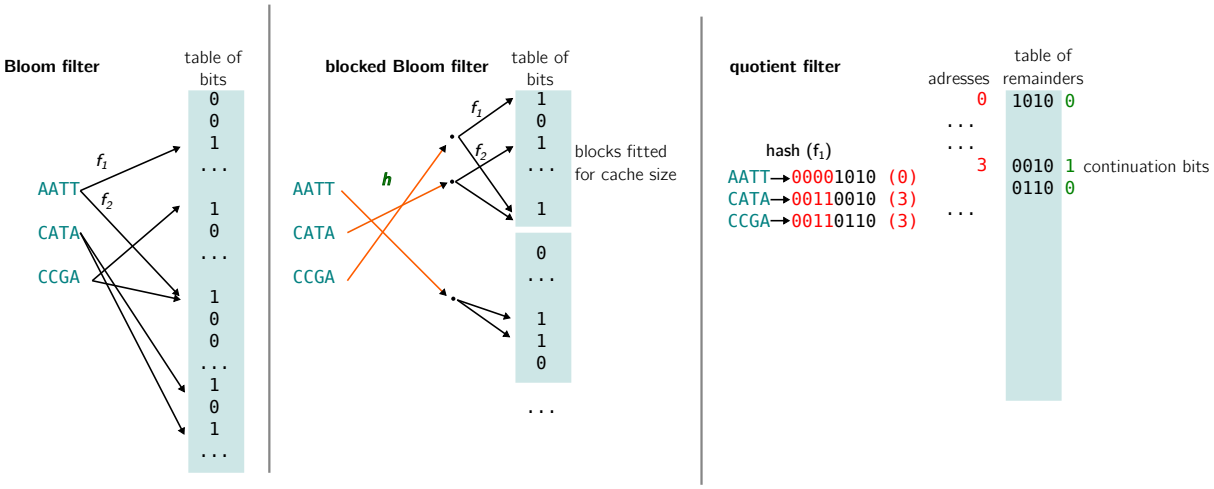

Figure 5 – Intuition of k-mer encoding with filters. K-mer are hashed for addressing. Left: in the Bloom filters, bits are set to 1 for the given addresses, with possible collisions (e.g.). Middle: the blocked Bloom filter divides a Bloom filter in blocks, where k-mers are addressed with a first hash function. Then blocks are filled as standard Bloom filters. Right: In quotient filter, the quotient (red part) of the hash addresses the remainder, and a probing strategy takes place when several remainders end up at the same location. In that example it is the case for CATA and CCGA here, a continuation bit indicates that there are several remainders to be checked. I chose to represent two hash functions for the Bloom filter, as in practice many tools use 1 to 3.

5.1.1. Bloom filters. Bloom filters are probably the most common probabilistic (yielding false positive) structure for k-mer sets.

Bloom filters are a space-efficient data structure that use a bit vector and multiple hash functions to represent a set of k-mers. During construction, each k-mer is hashed, and the corresponding bits in the bit vector are set to 1. To check if a k-mer is in the set, the same hash functions are applied, and if all corresponding bits are 1, the k-mer is considered to be present. However, collisions (different k-mers hashing to the same bits) can lead to false positives (this type of structure is sometimes called *approximate membership query* or AMQ) (see Figure 5 left).

Bloom filters are used because the theoretical space that has to be reserved to store any possible k-mer (4^k) using a bit set far exceeds practical datasets. For example, a set of 10 billion k-mers (even in large metagenomics datasets) is orders of magnitude smaller than 4^{31}. Therefore, they aim to project k-mers into a constrained space, hashing them into integers within a range (e.g., 0 to $2^{32} - 1$). They use hashing to uniformly distribute k-mers, avoiding dense and sparse regions in the filter. Using Bloom filters and a high rate of false positives ($> 0.1\%$) can lead to representing 31-mers of the human genome below 5 GB.

Bloom filters are straightforward to implement and, to some extent, dynamic. However, over time, the bit array can become saturated with 1s, necessitating resizing. The query is also not as fast as other methods based on hashing. When k-mers are the input, Bloom filters can benefit from highly efficient construction with Kmtricks Lemane et al., 2022.
Some works aimed to restore locality properties lost in hashing, such as blocked Bloom filters [Putze et al., 2010] or interleaved Bloom filters [Dadi et al., 2018]. Blocked Bloom filters place k-mers in in blocks within a bit array, and these blocks fit the cache (typically on 64 bits), so queried k-mers require loading a single location. This strategy improves the query but require $\sim$ 30% more space (Figure 5).

There exist variations to Bloom filters, such as the counting Bloom filter that registers approximate counts instead of presence/absence [Fan et al., 2000]. Related to our survey, recent improvement involving Bloom filters aim at representing abundances associated to k-mers in a compressed way [Shibuya et al., 2022]. Other works targeted the reduction of false positives in Bloom filters by indexing storing the list of s-mers ($s < k$) composing a k-mer instead of the k-mer itself [Robidou and Peterlongo, 2021, 2023].

The state of the art for this type of filters is XOR [Graf and Lemire, 2020] and Fuse filters [Graf and Lemire, 2022], with yet no many contributions dedicated to k-mers, e.g. this exception [Ulrich and Renard, 2024] in taxonomic assignment.

5.1.2. Other filters.

Quotient filters. are another structure used for approximate set membership tests, similar to Bloom filters. It provides efficient insert, delete, and membership query operations with controlled false positive rates and space efficiency. Quotient filters leverage quotienting (see Figure 5 right): the hash value is split in two parts, high order (prefix, or quotient) and low order (suffix, or remainder) bits. Quotient bits address k-mers in the structure, while remainders are stored as fingerprints. The filter uses an array of buckets, where each bucket can store multiple entries. Quotient filters use less space than traditional hash tables and can be more space-efficient than Bloom filters in certain scenarios. They support dynamic operations (insertion and deletion) more seamlessly than Bloom filters. Recent advances in quotient filters mitigate performance degradation as they fill up [Pandey et al., 2021], enhancing large scale use, or specialize in associating counts to k-mers [Levallois et al., 2024; Pandey et al., 2017a, 2018] and associated to a de Bruijn graph [Pandey et al., 2017b].

Cuckoo filters. stores fingerprints of elements in a table with multiple possible locations, which reduces the risk of collisions. If a collision occurs, the element can "kick out" (as cuckoos do) an existing element to another location, making cuckoo filters more space-efficient than Bloom filters. In Zentgraf et al., 2020, an advanced type of cuckoo filters is chosen because it maintains cache efficiency when dealing with gapped k-mers, which are k-mers separated by a fixed number of nucleotides (gaps) rather than being consecutive.

Example use cases of filters. Bloom filters are spread in a very large number of bioinformatics tools, often used for k-mer pre-filtering. Interestingly, some tools can fully handle the false positives they produce and end up being exact, such as the assembler Minia that detects the false positives thanks to the de Bruijn graph structure, or khmer [Crusoe et al., 2015]. Bifrost [Holley and Melsted, 2019] builds unitigs and uses Bloom filters for speed-up and corrects a posteriori

erroneous k-mers. Aside from assembly, Minia was used in [Krannich et al., 2022], a study involves analyzing genomic sequences from diverse human populations to detect non-reference variants absent from the human reference GRCh38.
A study on Ossabaw minipigs, that are genetically predisposed to a metabolic syndrome that increase the risk of heart disease, stroke, and diabetes, used a k-mer approach based on cuckoo filters that identified differences in clusters of genes encoding mitochondrial and inflammatory proteins [Kleinbongard et al., 2022].

5.2. Hash tables

5.2.1. Dynamic hash tables. Hash tables are dictionaries associating keys and values (e.g., genomes of origin) using hash functions. Dictionaries typically store their key sets and the associated values. In our case, a k-mer is hashed using a hash function, which provides an address in the hash table where it handled to be stored as a key with its associated value.

General purpose hash tables are used for k-mers whenever dynamicity is required and the cost in bit per k-mer is not a bottleneck. Among very competitive options, Rust's HashSet is based on state-of-the-art Swiss Tables, and other possibilities and their different trade-offs benefit from a comprehensive benchmark[2]. Díaz-Domínguez et al., 2024 present a space-efficient method for counting k-mers that involves a k-mer hash table. It presents similarity with SPSS and also the SBWT approach, as it uses k-1 overlaps between consecutive k-mers and records only some of the de Bruijn graph edges in its structure. Almost each entry k-mer only records its last symbol and a pointer to the previous k-mer, which allows to quickly retrieve and update counts for repeated patterns in the data.

5.2.2. Static hash tables.

Minimal perfect hash functions. Efficient hash functions (minimal perfect hash functions, MPHF) have been developed to associate keys with values. Contrary to regular hash functions MPHFs allocate space exactly for the required number of distinct k-mers. Regular hash tables, on the contrary, are dynamic and allocate space according to a desired load factor (Figure 7). However MPHFs cannot handle alien keys once built for a given set, which means that MPHFS coupled with a system to store exact keys can lead to hash tables, while MPHFs coupled with lossy fingerprints have a filter behavior (see for instance [Yu et al., 2018a]).

When MPHFs help create hash tables, such as BLight [Marchet et al., 2021] or SSHash [Pibiri, 2022], they are capable of k-mer presence/absence queries and associating additional information with k-mers.

In these hash tables, the second key ingredient is the technique to store the key set. One drawback of the MPHFs is that they must be fed a set of distinct k-mers, they cannot handle a regular FASTA. In order to leverage the redundancy in thes k-mer set to be stored and to provide ak-mer set, SPSS can be used. They have the property to store keys compactly. One of the first examples leveraging this principle is the Pufferfish k-mer index Almodaresi et al., 2018, using unitigs and a MPHF. Most efficient hash tables based on MPHF achieve storing 31-mers of a human genome in around 2-3 GB.

Minimizer partitioning. Minimizers [Roberts et al., 2004; Schleimer et al., 2003] are the product of selecting the smallest m-mer within a k-mer. Several interesting properties ensue, such as the fact that consecutive k-mers are likely to share a minimizer. Practically, minimizers are not

[2] https://github.com/martinus/map_benchmark

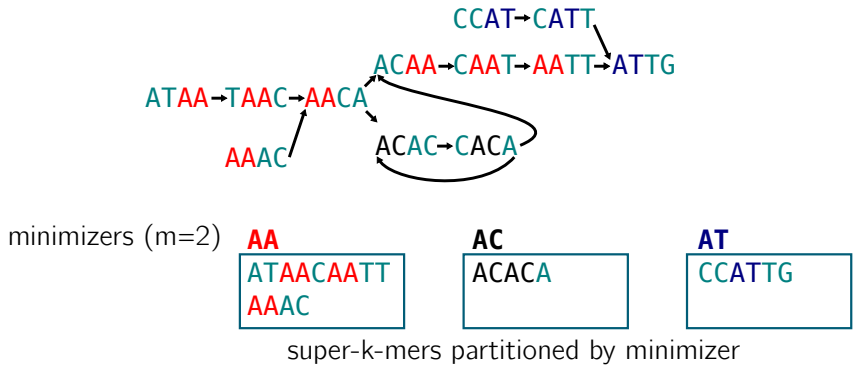

Figure 6 – Example of super-k-mers built with a lexicographical minimizer of size 2.

selected on a lexicographic order, but by hashing m-mers and selecting the smallest integer. This has empirically proven to have better distribution properties [Chikhi et al., 2014] for the minimizers. Minimizer induce a natural and deterministic partition of the k-mers, by grouping k-mers in 4^m buckets according to their minimizer. This property is helpful to reduce encoding integer sizes and parallelism. They are used coupled to MPHFs for instance in [Chikhi et al., 2016; Pibiri, 2022].

Super-*k*-mers. Tools like BLight utilize a special form of SPSS, the super-k-mers. These super-k-mers group consecutive k-mers and naturally partition k-mer sets, facilitating smaller integer usage and parallelization. They are an interesting case of SPSS, as they are driven by hashing techniques: super-k-mers are based on hashed minimizers (Figure 6).
Since consecutive k-mers stay close in the structure, this has positive impact on the query speed, as groups of k-mers in the query are queried almost simultaneously. It is worth mentioning that super-k-mers can be built from reads, unitigs, or other longer strings. Other properties for compression emerge when several samples are stored and compressed together using a colored de Bruijn graph, because consecutive k-mers are likely to share similar information [Karasikov et al., 2020; Pibiri et al., 2024].

Example use cases of hash tables. Hash tables are also a widespread data structure across bioinformatics. They are a method of choice for many unitig builders, such as BCALM2 [Chikhi et al., 2016] or Cuttlefish2 [Khan et al., 2022]. They are oftentimes embedded in colored de Bruijn graphs for joint k-mer analysis of multiple samples such as in [Fan et al., 2024; Marchet et al., 2020], but also in alignment-free methods [Almodaresi et al., 2021], and alignment free methods for RNA-seq quantification [Patro et al., 2017b] (using a hash table on the cuckoo principle).

The minimizer partitioning technique spread across sequence bioinformatics, it is for instance used in k-mer based metagenomic classifier Kraken2 [Wood et al., 2019], Kmtricks Lemane et al., 2022 or in the Kmer File Format (KFF) k-mer manager [Dufresne et al., 2022]. Associated to super-k-mers built from reads, it is also behind the partitioning of the KMC2Deorowicz et al., 2015 k-mer counter, that uses a disk-based sort-and-merge paradigm.

6. Conclusion

6.1. Summary

The impressive advancements in sequencing technologies are often rightfully praised, including in popular science. However, k-mer-based methods, which are crucial for effectively handling

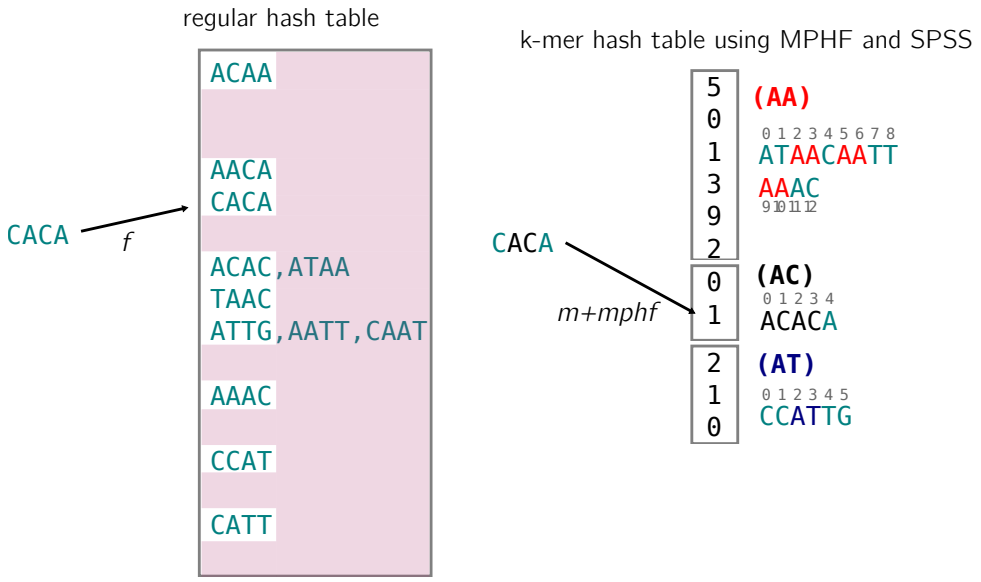

Figure 7 – Comparison of a regular hash table and a k-mer hash table using a MPHF. In the regular hash table on the left, bits allocated for the red part participate in a larger overhead than for MPHF-based methods. They ensure the possibility to add new elements while keeping an efficient query, and manage collisions. Keys are stored independently. On the left, a k-mer is addressed to a bucket thanks to its minimizer (here lexicographic, of size 2), and to store the key, a MPHF associates the k-mer to its position in a SPSS, so several keys are co-encoded.

this data, are frequently dismissed as mere technical details. These methods, however, represent significant achievements, combining advanced algorithms and excellent software engineering. They enable us to store the entire human genome's sequence in just a few gigabytes of RAM—a capacity that can fit into a modern smartphone.

In the current state of the art, I identify two main strategies for k-mer set representation (Figure 8):

- ◇ Storing k-mer fingerprints in slots, with space allocated closely matching the real number of k-mers, using small fingerprints with potential false positives (filters) or co-encoded (hash tables based on SPSS).
- ◇ Using lexicographic information of the k-mers or the graph, with transforms or algorithms that re-arranges the k-mer information in order to reduce the size of the representation.
- ◇ These two strategies, or sub-strategies, can be mixed. Typically, entry k-mers are treated as hashes in hash tables but efficient hash tables store their keys in compact way using lexicographic properties. Another example is is the super-k-mers, that leverage hash functions to create k-mer superstrings (compact lexicographic representations) for efficient k-mer partitioning.

Ten years back, mostly probabilistic methods could allow going below the information-theory space lower bound, with a few exact methods, at the price of offering a shallow range of operations. Nowadays, when solely looking at the bit per k-mer footprint, recent most well-performing methods are neck and neck and below the lower bound, and provide membership and associativity. Single queries are in the order of hundreds of nanoseconds.

Key differences include that SPSS-based solutions preserve partial k-mer order, an important property for quick queries in hash tables and compression in colored structures. BWT-based

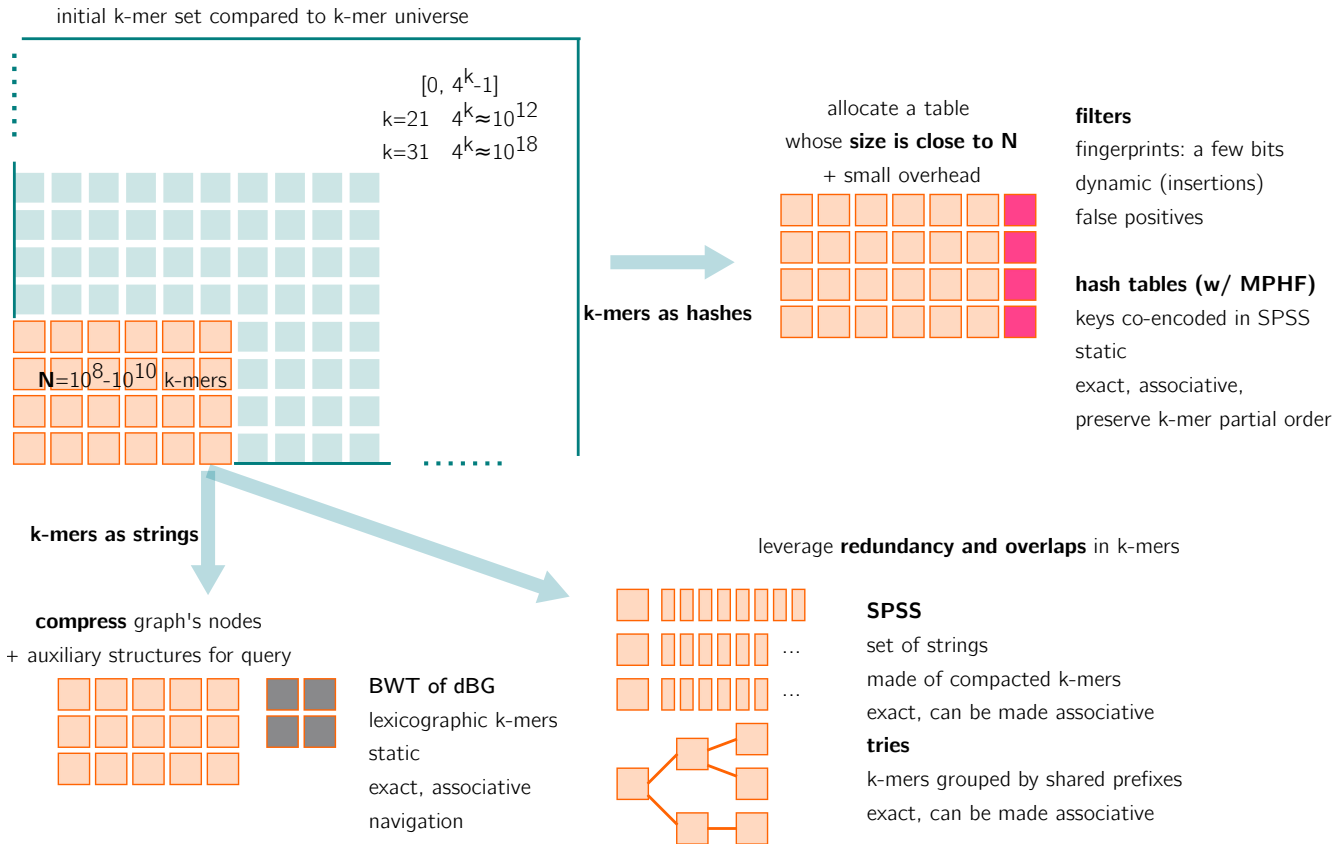

Figure 8 – Summary of methodological choices in k-mer sets representations

methods provide associative, highly compressed structures yielding k-mer ranks. Filters and probabilistic methods are still being used because they allow some dynamicity (k-mer insertion) and are especially quick to build. However, recent advances offer fully dynamic and exact solutions using slightly more space.

About practical tool usage, a confusing aspect is that many methods call themselves de Bruijn graphs but do not actually output sequences of a de Bruijn graph. This is because they rather consider the de Bruijn graph as an inner representation for the k-mer sets than about a final product. I give a summary in Figure 9. Not mentioned in details, some k-mer tools act as k-mer managers (writing k-mers on disk, partitioning k-mers ...) such as [Crusoe et al., 2015; Dufresne et al., 2022].

6.2. Trends and future directions for k-mer sets

Are we hitting a wall in terms of efficiency?. We don't have many informative lower bounds, which makes it difficult to answer. The community is still exploring, notably by discovering new structural properties of k-mers and sets [Abrar and Medvedev, 2024]. Currently, mostly lexicographic transforms based on rotations are used, but other approaches could be tested for better representation. Another option, learned indexes [Ferragina and Vinciguerra, 2020], is being explored to capture structural properties for optimized representation and queries. They can also be used for compression, especially for unassembled data for which there is room for improvement. Then, other techniques from neighbour computer science fields can inspire techniques close to minimal perfect hash functions (MPHFs) and Bloom filters, that could be expanded to other techniques close to filters and MPHF (such as XOR and Fuse filters, Othello [Graf and

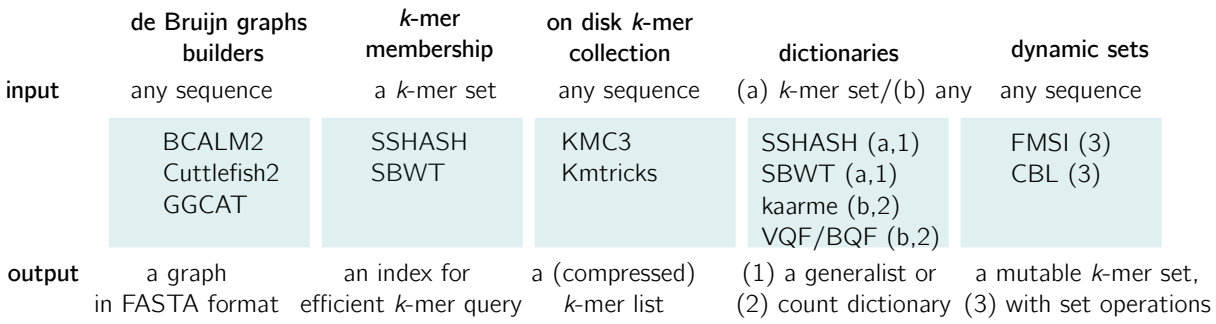

Figure 9 – Some k-mer structures seen through their functional aspects. Among tools that specialize in building de Bruijn graph sequences, BCALM2 focuses on being memory lightweight, Cuttelfish2 on scaling to large size of inputs, and GGCAT on speed. BCALM2 and Cuttelfish output a unitig graph, GGCAT has several SPSS options. BCALM2 and Cuttlefish yield a graph written on disk, GGCAT allows to load it in RAM for queries. Cuttlefish (and also TwoPaCo, have an option to speed-up construction if sequences are an assemble genome). K-mer indexes, dictionaries and sets can be loaded in RAM for membership queries. In k-mer indexes, SSHash is memory efficient and focuses on query speed, and SBWT provides compression.

Lemire, 2020, 2022; Yu et al., 2018b]). In order to compare all those techniques and the future ones, a comprehensive benchmark comparing performance in terms of construction time, query time, and disk/RAM usage, based on the size of the initial set and the value of k, is still lacking.

Good *k*-mer partitions.. More technical but critical for performances, the improvement of the computation of minimizers and super-k-mers is also actively researched [Hoang et al., 2024; Koerkamp and Pibiri, 2024; Kunzmann, 2024]. Recently, it lead to proposing the first minimal perfect hash function dedicated to k-mers [Pibiri et al., 2023], that can go below the space lower bound of general-purpose MPHF in certain scenarios. One interesting direction is k-mer partitioning using minimizers. Assigning a k-mer to its minimizer is a natural partition, but in practice, it is not very uniform in the target space, even when using hashing. This results in additional resource costs for large datasets. The solutions found are generally relegated to appendices and are minimally or not evaluated. We still lack a more theoretical framework on this issue.

Query driven developments. Using hashing is essential to fill structures that allocate tables or bit arrays. However, hashing loses the lexicographic information that k-mers carry. Not only two consecutive k-mers are typically parted, but it makes queries less informative in terms of distance between the sequence and the content of the index. Several hybrids (hash+string representation of k-mers, trie+quotienting, ...) try to mitigate the problem, but some are to my knowledge not yet explored, such as Bloom filters + SPSS. Another promising approach is to record partial lexicographic information in fingerprints that have properties close to hash [Greenberg et al., 2023; Shen and Iqbal, 2024]. It is especially interesting coupled to applications where k-mers are sampled and very sparse, where string-based approaches are less helpful.

Longer *k*-mers and different alphabets. The field is partially shaped by the type of sequencing data that becomes dominantly used and indexed. The growing throughput of sequencing data also motivates the possibility to index k-mers in streaming, therefore operations are expanding to include insertions and set operations. Long reads also become more and more accurate, which should motivate algorithmic solutions for very large k-mers in the future, as well as extended alphabet supporting IUPAC alphabets, since more and more modified bases are being called in reads such as from Oxford Nanopore.

***K*-mer based tools for bioinformatics analysis**. Finally, in a companion paper [Marchet, 2024], I review one direct application for k-mer set structures: structures than aggregate different sets from multiple datasets or genomes, called colored k-mer sets.

Fundings

This study has been supported by ANR JCJC Find-RNA [ANR -23-CE45-0003-01].

Conflict of interest disclosure

The author declares that she complies with the PCI rule of having no financial conflicts of interest in relation to the content of the article.

Data, script, code, and supplementary information availability

None declared.

Acknowledgments

This manuscript and its companion were created using my notes from talks I gave in recent conferences, courses and workshops. I'd like to thank the community for inviting me and giving me a opportunity to present and discuss my views on those subjects.

References

Abrar MH, Medvedev P (2024). *PLA-complexity of k-mer multisets*. *bioRxiv*. https://doi.org/10.1101/2024.02.08.579510. eprint: https://www.biorxiv.org/content/early/2024/02/11/2024.02.08.579510.full.pdf. URL: https://www.biorxiv.org/content/early/2024/02/11/2024.02.08.579510.

Agret C, Chateau A, Droc G, Sarah G, Ruiz M, Mancheron A (2022). *RedOak: a reference-free and alignment-free structure for indexing a collection of similar genomes*. *Journal of Open Source Software* **7**, 4363. https://doi.org/10.21105/joss.04363. URL: https://doi.org/10.21105/joss.04363.

Alanko J, Alipanahi B, Settle J, Boucher C, Gagie T (2021). *Buffering updates enables efficient dynamic de Bruijn graphs*. *Computational and structural biotechnology journal* **19**, 4067–4078.

Alanko JN, Biagi E, Puglisi SJ (2023a). *Longest common prefix arrays for succinct k-spectra*. In: *International Symposium on String Processing and Information Retrieval*. Springer, pp. 1–13.

Alanko JN, Puglisi SJ, Vuohtoniemi J (2023b). *Small searchable κ-spectra via subset rank queries on the spectral burrows-wheeler transform*. In: *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA23)*. SIAM, pp. 225–236.

Alipanahi B, Kuhnle A, Puglisi SJ, Salmela L, Boucher C (2021). *Succinct dynamic de Bruijn graphs*. *Bioinformatics* **37**, 1946–1952.

Alipanahi B, Muggli MD, Jundi M, Noyes NR, Boucher C (2020). *Metagenome SNP calling via read-colored de Bruijn graphs*. *Bioinformatics* **36**, 5275–5281.

Almodaresi F, Sarkar H, Srivastava A, Patro R (2018). *A space and time-efficient index for the compacted colored de Bruijn graph*. *Bioinformatics* **34**, i169–i177.

Almodaresi F, Zakeri M, Patro R (2021). *PuffAligner: a fast, efficient and accurate aligner based on the Pufferfish index*. *Bioinformatics* **37**, 4048–4055.

Bankevich A, Nurk S, Antipov D, Gurevich AA, Dvorkin M, Kulikov AS, Lesin VM, Nikolenko SI, Pham S, Prjibelski AD, et al. (2012). *SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing*. *Journal of computational biology* **19**, 455–477.

Bonin N, Doster E, Worley H, Pinnell LJ, Bravo JE, Ferm P, Marini S, Prosperi M, Noyes N, Morley PS, et al. (2023). *MEGARes and AMR++, v3. 0: an updated comprehensive database of antimicrobial resistance determinants and an improved software pipeline for classification using high-throughput sequencing*. *Nucleic acids research* **51**, D744–D752.

Bowe A, Onodera T, Sadakane K, Shibuya T (2012). *Succinct de Bruijn Graphs*. In: *Algorithms in Bioinformatics*. Ed. by Ben Raphael and Jijun Tang. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 225–235.

Brinda K (2016). *Nouvelles techniques informatiques pour la localisation et la classification de données de séquençage haut débit*. PhD thesis. Paris Est.

Břinda K, Baym M, Kucherov G (2021). *Simplitigs as an efficient and scalable representation of de Bruijn graphs*. *Genome biology* **22**, 1–24.

Bushmanova E, Antipov D, Lapidus A, Prjibelski AD (2019). *rnaSPAdes: a de novo transcriptome assembler and its application to RNA-Seq data*. *GigaScience* **8**, giz100.

Chikhi R (2021). *A tale of optimizing the space taken by de Bruijn graphs*. In: *Connecting with Computability: 17th Conference on Computability in Europe, CiE 2021, Virtual Event, Ghent, July 5–9, 2021, Proceedings 17*. Springer, pp. 120–134.

Chikhi R, Holub J, Medvedev P (2019). *Data Structures to Represent a Set of k-long DNA Sequences*. *ACM Computing Surveys* **54**, 17:1–17:22.

Chikhi R, Limasset A, Jackman S, Simpson JT, Medvedev P (2014). *On the Representation of de Bruijn Graphs*. In: *Research in Computational Molecular Biology*. Ed. by Roded Sharan. Cham: Springer International Publishing, pp. 35–55.

Chikhi R, Limasset A, Medvedev P (2016). *Compacting de Bruijn graphs from sequencing data quickly and in low memory*. *Bioinformatics* **32**, i201–i208. https://doi.org/10.1093/bioinformatics/btw279. eprint: http://oup.prod.sis.lan/bioinformatics/article-pdf/32/12/i201/17130531/btw279.pdf. URL: https://doi.org/10.1093/bioinformatics/btw279.

Conway TC, Bromage AJ (2011). *Succinct data structures for assembling large genomes*. *Bioinformatics* **27**, 479–486.

Cracco A, Tomescu AI (2023). *Extremely fast construction and querying of compacted and colored de Bruijn graphs with GGCAT*. *Genome Research*, gr–277615.

Crusoe MR, Alameldin HF, Awad S, Boucher E, Caldwell A, Cartwright R, Charbonneau A, Constantinides B, Edvenson G, Fay S, et al. (2015). *The khmer software package: enabling efficient nucleotide sequence analysis*. *F1000Research* **4**.

Dadi TH, Siragusa E, Piro VC, Andrusch A, Seiler E, Renard BY, Reinert K (2018). *DREAM-Yara: an exact read mapper for very large databases with short update time*. *Bioinformatics* **34**, i766–i772.

Deorowicz S, Kokot M, Grabowski S, Debudaj-Grabysz A (2015). *KMC 2: fast and resource-frugal k-mer counting*. *Bioinformatics* **31**, 1569–1576.

Díaz-Domínguez D, Leinonen M, Salmela L (2024). *Space-efficient computation of k-mer dictionaries for large values of k*. *Algorithms for Molecular Biology* **19**, 14.

Dufresne Y, Lemane T, Marijon P, Peterlongo P, Rahman A, Kokot M, Medvedev P, Deorowicz S, Chikhi R (2022). *The K-mer File Format: a standardized and compact disk representation of sets of k-mers*. *Bioinformatics* **38**, 4423–4425.

Fan J, Khan J, Singh N, et al. (2024). *Fulgor: a fast and compact k-mer index for large-scale matching and color queries*. *Algorithms for Molecular Biology* **19**, 3. https://doi.org/10.1186/s13015-024-00251-9. URL: https://doi.org/10.1186/s13015-024-00251-9.

Fan L, Cao P, Almeida J, Broder AZ (2000). *Summary cache: a scalable wide-area web cache sharing protocol*. *IEEE/ACM Transactions on Networking (TON)* **8**, 281–293. https://doi.org/10.1109/90.851975.

Ferragina P, Manzini G (2005). *Indexing compressed text*. *J. ACM* **52**, 552–581.

Ferragina P, Vinciguerra G (2020). *The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds*. *Proceedings of the VLDB Endowment* **13**, 1162–1175.

Graf TM, Lemire D (2020). *Xor filters: Faster and smaller than bloom and cuckoo filters*. *Journal of Experimental Algorithmics (JEA)* **25**, 1–16.

Graf TM, Lemire D (2022). *Binary fuse filters: Fast and smaller than xor filters*. *Journal of Experimental Algorithmics (JEA)* **27**, 1–15.

Greenberg G, Ravi AN, Shomorony I (2023). *LexicHash: sequence similarity estimation via lexicographic comparison of hashes*. *Bioinformatics* **39**, btad652.

Hannoush K, Marchet C, Peterlongo P (2024). *Cdbgtricks: strategies to update a compacted de Bruijn graph*. In: *Proceedings of the Prague Stringology Conference 2024 (PSC 2024)*. Czech Technical University in Prague, pp. 202–205. URL: https://psc.fit.cvut.cz/event/2024/.

Hoang M, Marçais G, Kingsford C (2024). *Density and Conservation Optimization of the Generalized Masked-Minimizer Sketching Scheme*. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology* **31**, 2–20. https://doi.org/10.1089/cmb.2023.0212.

Holley G, Melsted P (2019). *Bifrost–Highly parallel construction and indexing of colored and compacted de Bruijn graphs*. *BioRxiv*, 695338.

Holley G, Wittler R, Stoye J (2016). *Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage*. *Algorithms for Molecular Biology* **11**, 3.

Ingels F, Martayan I, Salson M, Marchet C (2024). *Constrained enumeration of k-mers from a collection of references with metadata*. *bioRxiv*. https://doi.org/10.1101/2024.05.26.595967. URL: https://doi.org/10.1101/2024.05.26.595967.

Iqbal Z, Caccamo M, Turner I, Flicek P, McVean G (2012). *De novo assembly and genotyping of variants using colored de Bruijn graphs*. *Nature genetics* **44**, 226.

Jenike KM, Campos-Domínguez L, Boddé M, Cerca J, Hodson CN, Schatz MC, Jaron KS (2024). *Guide to k-mer approaches for genomics across the tree of life*. *arXiv preprint arXiv:2404.01519*. https://doi.org/10.48550/arXiv.2404.01519.

Karasikov M, Mustafa H, Danciu D, Zimmermann M, Barber C, Rätsch G, Kahles A (2020). *MetaGraph: Indexing and Analysing Nucleotide Archives at Petabase-scale*. *bioRxiv*. https://doi.org/10.1101/2020.10.01.322164. eprint: https://www.biorxiv.org/content/early/2020/11/03/2020.10.01.322164.full.pdf. URL: https://www.biorxiv.org/content/early/2020/11/03/2020.10.01.322164.

Khan J, Kokot M, Deorowicz S, Patro R (2022). *Scalable, ultra-fast, and low-memory construction of compacted de Bruijn graphs with Cuttlefish 2*. *Genome Biology* **23**, 190. https://doi.org/10.1186/s13059-022-02743-6.

Kleinbongard P, Lieder HR, Skyschally A, Alloosh M, Gödecke A, Rahmann S, Sturek M, Heusch G (2022). *Non-responsiveness to cardioprotection by ischaemic preconditioning in Ossabaw minipigs with genetic predisposition to, but without the phenotype of the metabolic syndrome*. *Basic research in cardiology* **117**, 58.

Koerkamp RG, Pibiri GE (2024). *The mod-minimizer: a simple and efficient sampling algorithm for long k-mers*. *bioRxiv*. https://doi.org/10.1101/2024.05.25.595898. eprint: https://www.biorxiv.org/content/early/2024/07/07/2024.05.25.595898.full.pdf. URL: https://www.biorxiv.org/content/early/2024/07/07/2024.05.25.595898.

Kokot M, Długosz M, Deorowicz S (2017). *KMC 3: counting and manipulating k-mer statistics*. *Bioinformatics* **33**, 2759–2761.

Krannich T, White WTJ, Niehus S, Holley G, Halldórsson BV, Kehr B (2022). *Population-scale detection of non-reference sequence variants using colored de Bruijn graphs*. *Bioinformatics* **38**, 604–611.

Kunzmann P (2024). *A fast and simple approach to k-mer decomposition*. *bioRxiv*, 2024–07.

Lemane T, Medvedev P, Chikhi R, Peterlongo P (2022). *kmtricks: efficient and flexible construction of Bloom filters for large sequencing data collections*. *Bioinformatics Advances* **2**, vbac029. https://doi.org/10.1093/bioadv/vbac029. eprint: https://academic.oup.com/bioinformaticsadvances/article-pdf/2/1/vbac029/47086128/vbac029.pdf. URL: https://doi.org/10.1093/bioadv/vbac029.

Levallois V, Andreace F, Le Gal B, Dufresne Y, Peterlongo P (2024). *The Backpack Quotient Filter: a dynamic and space-efficient data structure for querying k-mers with abundance*. *bioRxiv*, 2024–02.

Mäklin T, Kallonen T, Alanko J, Samuelsen Ø, Hegstad K, Mäkinen V, Corander J, Heinz E, Honkela A (2021). *Bacterial genomic epidemiology with mixed samples*. *Microbial genomics* **7**, 000691.

Marcais G, Kingsford C (2012). *Jellyfish: A fast k-mer counter*. *Tutorialis e Manuais* **1**, 1038.

Marchet C (2024). *Advancements in Colored k-mer Sets: Essentials for the Curious*. *aRxiv*. Preprint. https://doi.org/tobeupdated.

Marchet C, Iqbal Z, Gautheret D, Salson M, Chikhi R (2020). *REINDEER: efficient indexing of k-mer presence and abundance in sequencing datasets*. *Bioinformatics* **36**, i177–i185.

Marchet C, Kerbiriou M, Limasset A (2021). *BLight: efficient exact associative structure for k-mers*. *Bioinformatics* **37**, 2858–2865.

Marini S, Mora RA, Boucher C, Robertson Noyes N, Prosperi M (2022). *Towards routine employment of computational tools for antimicrobial resistance determination via high-throughput sequencing*. *Briefings in bioinformatics* **23**, bbac020.

Martayan I, Cazaux B, Limasset A, Marchet C (2024). *Conway-Bromage-Lyndon (CBL): an exact, dynamic representation of k-mer sets*. *bioRxiv*, 2024–01.

Minkin I, Pham S, Medvedev P (2016). *TwoPaCo: an efficient algorithm to build the compacted de Bruijn graph from many complete genomes*. *Bioinformatics* **33**, 4024–4032. https://doi.org/10.1093/bioinformatics/btw609. eprint: https://academic.oup.com/bioinformatics/article-pdf/33/24/4024/49042096/bioinformatics_33_24_4024.pdf. URL: https://doi.org/10.1093/bioinformatics/btw609.

Nguyen HT, Xue H, Firlej V, Ponty Y, Gallopin M, Gautheret D (2021). *Reference-free transcriptome signatures for prostate cancer prognosis*. *BMC cancer* **21**, 1–12.

Pandey P, Bender MA, Johnson R, Patro R (2017a). *A general-purpose counting filter: Making every bit count*. In: *Proceedings of the 2017 ACM international conference on Management of Data*, pp. 775–787.

Pandey P, Bender MA, Johnson R, Patro R (2018). *Squeakr: an exact and approximate k-mer counting system*. *Bioinformatics* **34**, 568–575.

Pandey P, Bender MA, Johnson R, Patwa S (2017b). *deBGR: an efficient and near-exact representation of the weighted de Bruijn graph*. *Bioinformatics* **33**, i133–i141. https://doi.org/10.1093/bioinformatics/btx261. URL: https://doi.org/10.1093/bioinformatics/btx261.

Pandey P, Conway A, Durie J, Bender MA, Farach-Colton M, Johnson R (2021). *Vector quotient filters: Overcoming the time/space trade-off in filter design*. In: *Proceedings of the 2021 International Conference on Management of Data*, pp. 1386–1399.

Patro R, Duggal G, Love MI, Irizarry RA, Kingsford C (2017a). *Salmon provides fast and bias-aware quantification of transcript expression*. *Nature methods* **14**, 417–419.

Patro R, Duggal G, Love MI, Irizarry RA, Kingsford C (2017b). *Salmon provides fast and bias-aware quantification of transcript expression*. *Nature methods* **14**, 417–419.

Pibiri GE, Fan J, Patro R (2024). *Meta-colored compacted de Bruijn graphs*. In: *International Conference on Research in Computational Molecular Biology*. Cham: Springer Nature Switzerland, pp. 131–146.

Pibiri GE (2022). *Sparse and skew hashing of k-mers*. *Bioinformatics* **38**, i185–i194.

Pibiri GE, Shibuya Y, Limasset A (2023). *Locality-preserving minimal perfect hashing of k-mers*. *Bioinformatics* **39**, i534–i543.

Putze F, Sanders P, Singler J (2010). *Cache-, hash-, and space-efficient bloom filters*. *Journal of Experimental Algorithmics (JEA)* **14**, 4–4.

Rahman A, Medevedev P (2021). *Representation of k-mer sets using spectrum-preserving string sets*. *Journal of Computational Biology* **28**, 381–394.

Rahman A, Medvedev P (2022). *Assembler artifacts include misassembly because of unsafe unitigs and underassembly because of bidirected graphs*. *Genome Research* **32**, 1746–1753.

Riquier S, Bessiere C, Guibert B, Bouge AL, Boureux A, Ruffle F, Audoux J, Gilbert N, Xue H, Gautheret D, et al. (2021). *Kmerator Suite: design of specific k-mer signatures and automatic metadata discovery in large RNA-seq datasets*. *NAR genomics and bioinformatics* **3**, lqab058.

Roberts M, Hayes W, Hunt BR, Mount SM, Yorke JA (2004). *Reducing storage requirements for biological sequence comparison*. *Bioinformatics* **20**, 3363–3369.

Robidou L, Peterlongo P (2021). *findere: fast and precise approximate membership query*. In: *String Processing and Information Retrieval: 28th International Symposium, SPIRE 2021, Lille, France, October 4–6, 2021, Proceedings 28*. Springer, pp. 151–163.

Robidou L, Peterlongo P (2023). *fimpera: drastic improvement of Approximate Membership Query data-structures with counts*. *Bioinformatics* **39**, btad305.

Rossignolo E, Comin M (2024). *Enhanced Compression of k-Mer Sets with Counters via de Bruijn Graphs*. *Journal of Computational Biology*. In press. https://doi.org/tobeupdated.

Sawada J, Williams A (2017). *Practical algorithms to rank necklaces, Lyndon words, and de Bruijn sequences*. *Journal of Discrete Algorithms* **43**, 95–110.

Schleimer S, Wilkerson DS, Aiken A (2003). *Winnowing: local algorithms for document fingerprinting*. In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 76–85.

Schmidt S, Alanko JN (2023). *Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time*. *Algorithms for Molecular Biology* **18**, 5.

Schmidt S, Khan S, Alanko J, Tomescu AI (2021). *Matchtigs: minimum plain text representation of kmer sets*. *bioRxiv*.

Shen W, Iqbal Z (2024). *LexicMap: efficient sequence alignment against millions of prokaryotic genomes*. *bioRxiv*. https://doi.org/10.1101/2024.08.30.610459. eprint: https://www.biorxiv.org/content/early/2024/08/31/2024.08.30.610459.full.pdf. URL: https://www.biorxiv.org/content/early/2024/08/31/2024.08.30.610459.

Shibuya Y, Belazzougui D, Kucherov G (2022). *Space-efficient representation of genomic k-mer count tables*. *Algorithms for Molecular Biology* **17**, 5.

Sladký O, Veselý P, Břinda K (2023). *Masked superstrings as a unified framework for textual k-mer set representations*. *bioRxiv*. https://doi.org/10.1101/2023.02.01.526717. eprint: https://www.biorxiv.org/content/early/2023/02/03/2023.02.01.526717.full.pdf. URL: https://www.biorxiv.org/content/early/2023/02/03/2023.02.01.526717.

Sladkỳ O, Veselỳ P, Břinda K (2024). *Function-Assigned Masked Superstrings as a Versatile and Compact Data Type for k-Mer Sets*. *bioRxiv*, 2024–03.

Ulrich JU, Renard BY (2024). *Fast and space-efficient taxonomic classification of long reads with hierarchical interleaved XOR filters*. *Genome Research*, gr–278623.

Wood DE, Lu J, Langmead B (2019). *Improved metagenomic analysis with Kraken 2*. *Genome biology* **20**, 1–13.

Wood DE, Salzberg SL (2014). *Kraken: ultrafast metagenomic sequence classification using exact alignments*. *Genome biology* **15**, R46.

Yu Y, Liu J, Liu X, Zhang Y, Magner E, Lehnert E, Qian C, Liu J (2018a). *SeqOthello: querying RNA-seq experiments at scale*. *Genome Biology* **19**, 167.

Yu Y, Belazzougui D, Qian C, Zhang Q (2018b). *Memory-efficient and ultra-fast network lookup and forwarding using Othello hashing*. *IEEE/ACM Transactions on Networking* **26**, 1151–1164.

Zentgraf J, Timm H, Rahmann S (2020). *Cost-optimal assignment of elements in genome-scale multiway bucketed Cuckoo hash tables*. In: *Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX)*. Ed. by Guy Blelloch and Irene Finocchi. SIAM, pp. 186–198.