Towards Efficient k-Mer Set Operations via Function-Assigned Masked Superstrings

Ondřej Sladký, Pavel Veselý, Karel Břinda

Towards Efficient k-Mer Set Operations via Function-Assigned Masked Superstrings

Ondřej Sladký
Computer Science Institute of Charles University, Prague, Czechia
ETH Zürich, Switzerland

Pavel Veselý
Computer Science Institute of Charles University, Prague, Czechia

Karel Břinda
Inria, Irisa, Univ. Rennes, 35042 Rennes, France

Abstract

The design of efficient dynamic data structures for large k-mer sets belongs to central challenges of sequence bioinformatics. Recent advances in compact k-mer set representations via simplitigs/Spectrum-Preserving String Sets, culminating with the masked superstring framework, have provided data structures of remarkable space efficiency for wide ranges of k-mer sets. However, the possibility to perform set operations remained limited due to the static nature of the underlying compact representations. Here, we develop f-masked superstrings, a concept combining masked superstrings with custom demasking functions f to enable efficient k-mer set operations via string concatenation. Combined with the FMSI index for masked superstrings, we obtain a memory-efficient k-mer index supporting set operations via Burrows-Wheeler Transform merging. The framework provides a promising theoretical solution to a pressing bioinformatics problem and highlights the potential of f-masked superstrings to become an elementary data type for k-mer sets.

2012 ACM Subject Classification Applied computing → Bioinformatics; Theory of computation → Pattern matching

Keywords and phrases k-mer sets, data structures, set operations, masked superstrings

Funding *Ondřej Sladký*: Supported by GA ČR project 22-22997S and ERC-CZ project LL2406 of the Ministry of Education of Czech Republic.
Pavel Veselý: Supported by GA ČR project 22-22997S, ERC-CZ project LL2406 of the Ministry of Education of Czech Republic, and Center for Foundations of Modern Computer Science (Charles Univ. project UNCE 24/SCI/008).
Karel Břinda: Supported by French National Research Agency (ANR) under Grant ANR-24-CE45-1226 for the REALL project.

1 Introduction

To store and analyze the vast volumes of DNA sequencing data [61], modern bioinformatics methods increasingly rely on k-mers to bypass the computationally expensive sequence alignment and mitigate data heterogeneity. k-mer-based methods are used in a range of applications, including large-scale data search [6, 8, 14, 33], metagenomic classification [15, 62], infectious disease diagnostics [9, 13], assembly evaluation [52], and transcript abundance quantification [10, 48]. All such applications depend on the efficiency of the underlying k-mer data structures.

The design of efficient k-mer data structures is a central challenge of contemporary sequence bioinformatics [18, 43]. We summarize the desired properties of an ideal data structure in Section 1.1, but the two main requirements are as follows. First, as even a single k-mer set can be large, potentially containing up to hundreds of billions distinct k-mers [33], the main challenge is to provide data structures that are efficient in space and time simultaneously. Second, as modern genomic databases undergo rapid development, thanks to their growing content and curation efforts, rapid and space-efficient updates across k-mer indexes are increasingly needed to avoid repetitive and costly index recomputations. This includes scenarios such as k-mer set operations across sets, and additions or removals of individual k-mers.

The space efficiency has been particularly well addressed using the (repetitive) Spectrum-Preserving String Sets, (r)SPSS, that exploit k-mer non-independence [17]. In particular, the key observation is that genomic k-mer sets can be typically represented by k-long substrings of a small number of (arbitrarily long) strings, which is the so-called *spectrum-like property (SLP)* [18]. Building on this observation, several textual representations of k-mer sets were proposed. The first ones were based on de Bruijn graphs, to which we jointly refer as *(repetitive) Spectrum Preserving String Sets* or *(r)SPSS* [11, 12, 19, 20, 51, 57, 58]. (r)SPSS are currently highly standard and many data structures for k-mer sets base upon them [2, 7, 45, 49, 55],

Masked superstrings (MS) have provided additional space gains [59], by the virtue of a better k-mer set compaction. The core improvement over (r)SPSS lies in modeling the structure of k-mer sets by overlap graphs instead of de Bruijn graphs, thus being able to exploit overlaps of *any* length. MS represent k-mer sets using an approximately shortest superstring of all k-mers and a binary mask to avoid false positive k-mers. MS generalize any existing (r)SPSS representation as these can always be encoded as MS, but provide further compression power, especially for sets without the SLP, such as those arising from sketching or subsampling. The resulting representation is well indexable using a technique called Masked Burrows Wheeler Transform [60], resulting in a k-mer data structure with $2 + o(1)$ bits per k-mer under the SLP.

However, the lack of dynamicity of both (r)SPSS and MS – and of their derived data structures – has been limiting their wider applicability. To the best of our knowledge, the only supported operations were union, either via merging (r)SPSS of several k-mer sets resulting in an rSPSS of their union, or by the Cdbg-Tricks [28] to calculate the union unitigs from unitigs of mutliple original k-mer sets. However, other operations than union, such as intersection or symmetric difference have never been considered.

Here, we develop a dynamic variant of masked superstrings (and thus also of (r)SPSS) called the *f-masked superstrings (f-MS)*. The key idea is to equip masked superstrings with so-called demasking functions f for more flexible mask interpretation (**Section 3**). When complemented with the concatenation operation, this provides support for any set

operation, including union, intersection, and symmetric or asymmetric difference on the represented k-mer sets, resulting in a complete algebraic type for k-mer sets (**Section 4**). We implement the whole approach in the FMSI index for masked superstrings [60] (**Section 5**), and demonstrate applicability of the concept on an example dataset (**Section 6**).

1.1 Problem formulation

Before delving into technical details of our contribution, we formulate properties of an ideal k-mer data structure. The desired data structure for representing a set of k-mers should support the following operations in a space- and time-efficient way:

(i) **k-mer set index construction**, with time complexity linear with the number of k-mers.
(ii) **k-mer membership queries**, with low bits-per-k-mer memory requirements, approaching 2 bits per distinct k-mer for datasets following the Spectrum-Like Property [18] and proportionally for more complex datasets.
(iii) **Set operations**, including union, intersection, difference, and symmetric difference.
(iv) **Single k-mer deletion and insertion,** or at least one of these.

In this paper, we make progress on efficient set operations while retaining high space efficiency for k-mer membership queries and other desirable properties of (r)SPSS- or MS-based data structures.

Naïve approaches for performing set operations. We note that one can add support for set operations to a static data structure in a straightforward way: One option is extracting the k-mer sets from the input indexes, performing the given operation with the sets, and computing the new index for the resulting set; however, this process requires substantial time and memory. Another option is to keep the indexes for input k-mer sets and process a k-mer query on the set resulting from the operation by asking each index for the presence/absence of the k-mer in each input set, which is however time and memory inefficient as all the indexes need to be loaded into memory, as also noted in [30]. Therefore, we seek to perform set operations without the costly operation of recomputing the index or making multiple queries to original indexes.

1.2 Related Work

Many works have recently focused on data structures for single k-mer sets and their collections; we refer to [18, 43] for recent surveys. Here, we primarily focus on those that offer some kind of dynamicity, i.e., an efficient support for set operations or, at least, insertions/deletions of individual k-mers. The recently introduced Conway-Bromage-Lyndon (CBL) structure [46] builds on the work of Conway and Bromage [21] on sparse bit-vector encodings and combines them with smallest cyclic rotations of k-mers (a.k.a. Lyndon words), which yields a dynamic and exact k-mer index supporting set operations, such as union, intersection, and difference, as well as insertions or deletions of k-mers. However, the memory requirements for processing queries on dataset satisfying the spectrum-like property are substantially worse than for other, static methods [46].

While, to the best of our knowledge, other k-mer indexes do not support efficient set operations such as the intersection or difference, other tools, including BufBoss [1], DynamicBoss [4], and FDBG [22] allow for efficient insertions and deletions of individual k-mers. Bifrost [29], VARI-merge [47], Metagraph [33], dynamic Mantis [5], or the very recent Cdbg-Tricks [28] support insertions but not deletions. Other data structures, such as COBS [6], RAMBO [27], or kmtricks [35] trade exactness for space compression, allowing a certain false probability rate. These employ variants of the Bloom filter that can also process

insertions and compute unions efficiently but other set operations such as the intersection are not directly possible. We note that there are many more highly efficient but static data structures for individual or multiple k-mer sets, e.g., [2, 3, 7, 24, 25, 29, 40, 44, 45, 49]. Finally, one can also use the textual representations of (r)SPSS, with efficient string indexes such as the FM index [26] or BWA [36–39], but this only yields static indexes.

Another line of work, e.g. [23, 32, 34, 41, 42, 52, 53], focused on k-mer counting, where we additionally require to compute the k-mer frequencies. Out of the many k-mer counters available, GenomeTester4 [32], KMC3 [34], or Meryl [52] support operations such as union, intersection, or difference on multisets. However, k-mer counters require substantially larger memory than the most efficient k-mer indexes or heavily utilize disk.

2 Preliminaries

Strings and k-mers. We use constant-size alphabets Σ, typically the nucleotide alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (unless stated otherwise). The set Σ^* contains all finite strings over Σ, with ϵ representing the empty string. For a given string $S \in \Sigma^*$, $|S|$ denotes its length, and $|S|_c$ the number of occurrences of the letter c in S. For two strings S and T, $S + T$ denotes their concatenation. A *k-mer* is a k-long string over Σ, and unless stated otherwise, we assume *canonical k-mers*, i.e., a k-mer and its reverse complement are considered equal. For a string S and a fixed length k, the k-mers *generated* by S are all k-long substrings of S, and similarly for a set of strings $\mathcal{R}$, they are those generated by the individual strings $S \in \mathcal{R}$.

(r)SPSS representations of k-mer sets. For a given k-mer set K (comprising k-mers of the same size k), a set of strings $\mathcal{R}$ (called *simplitigs*) constitutes its *Spectrum Preserving String Set* (SPSS) [11, 12, 50, 51] if (1) it generates K and (2) each k-mer occurs only once among the simplitigs (uniqueness). Repetitive Spectrum Preserving String Sets (rSPSS) [56, 58], including its members called *matchtigs*, are defined similarly, except that condition (2) is omitted. To refer to either of these concepts, we use the abbreviation (r)SPSS .

Masked superstrings. Given a k-mer set K, a *masked superstring* (MS) [59] consists of a pair (S, M), where S is an arbitrary superstring of the k-mers in K and M is a binary mask of the same length. An occurrence of a k-mer in an MS is said to be ON if there is `1` at the corresponding position in the mask (i.e., the initial position of the occurrence), and OFF otherwise. The set of k-mers generated by S are referred to as the *appearing k-mers*, and are further classified based on their values in the mask into *represented k-mers* (at least one ON occurrence) and *ghost k-mers* (all occurrences OFF). All masks M that represent a given K in a combination with a given superstring are called *compatible*.

Encoding conventions. To implicitly encode k in MS, the last $k-1$ positions of masks are always set to `0` and preceded by `1` (always feasible). In this paper, MS are presented using their mask-case encoding, i.e., the superstring with individual letters in either lower or upper case indicating a `1` or `0` in the mask, respectively.

▶ **Example 1.** Consider the k-mer set $K = \{\mathtt{ACG}, \mathtt{GGG}\}$. One possible superstring is `ACGGGG`, with three compatible masks: `101100`, `100100`, `101000`, resulting in the encodings `AcGGgg`, `AcgGgg`, `AcGggg`, respectively. The latter mask would contravene our encoding assumptions, and thus, would not be used for MS storage. When parsing `AcgGgg`, the suffix implies $k = 3$, and the k-mers are decoded as $\{\mathtt{ACG}, \mathtt{GGG}\}$.

3 Function-Assigned Masked Superstrings

Let (M, S) be an MS (Masked Superstring) and suppose that our objective is to determine whether a given k-mer Q is among the MS-represented k-mers. Conceptually, this process consists of two steps: (1) identify the occurrences of Q in S, and (2) verify using the mask M whether at least one occurrence of Q is ON. We can formalize this process via a so-called *occurrence function*.

▶ **Definition 2.** *For a superstring S, a mask M, and a k-mer Q, the* occurrence function $\lambda(S, M, Q) \to \{0,1\}^*$ *is a function returning a finite binary sequence with the mask symbols of the corresponding occurrences, i.e.,*

$$\lambda(S, M, Q) := (M_i \,|\, S_i \cdots S_{i+k-1} = Q) \ . \tag{1}$$

In this notation, verifying k-mer presence corresponds to evaluating the composite function '$\mathbf{or} \circ \lambda$'; that is, a k-mer is present if $\lambda(S, M, Q)$ is non-empty and the logical **or** operation on these values yields 1. For instance, for `AcgGgg` from Example 1, and query k-mer $Q =$ `GGG`, it holds that $\lambda(S, M, Q) = (0, 1)$, as the first occurrence is OFF and the second is ON, with the **or** of these values being 1; therefore, `GGG` is represented.

The set of all MS-represented k-mers can thus be expressed as

$$K = \{Q \in \Sigma^k \,|\, \mathbf{or}(\lambda(S, M, Q)) = 1\}. \tag{2}$$

The key observation of this work is that **or** is only one member of a large class of possible "demasking functions" (see examples in Table 1). For instance, MS could have been defined using the **xor** function, with a k-mer considered present if the number of its ON occurrences is odd. Indeed, k-mer demasking can use any symmetric Boolean function, which we further equip with a special return value, `invalid`, as a means to impose criteria on mask validity.

▶ **Definition 3.** *We call a symmetric function $f : \{0,1\}^* \to \{0, 1, \texttt{invalid}\}$ a* k-mer demasking function.

In addition, we will typically require our demasking functions to have several natural properties. First, the non-appearing k-mers should be treated as non-represented (P1) (unless we aim to compactly represent set complements). Second, all appearing k-mers should be encodable using a compatible mask as present or absent, irrespective of the number of their occurrences (P2). Third, a single k-mer occurrence that is also ON should be interpreted as k-mer presence (P3). Fourth, the function should be efficiently computable from the frequencies of `0`s and `1`s (P4).

▶ **Definition 4.** *A demasking function f is* comprehensive *if it satisfies the following four properties:*

(P1) $f(\epsilon) = 0$.
(P2) For every $n > 0$, exist $x, y \in \{0,1\}^n$ such that $f(x) = 0$ and $f(y) = 1$.
(P3) $f((1)) = 1$ *and* $f((0)) = 0$.
(P4) Given $|x|_0$ and $|x|_1$, one can evaluate $f(x)$ in constant time in the wordRAM model.

With the notation of demasking functions f, we can now generalize the concept of MS to so-called f-masked superstrings (f-MS).

▶ **Definition 5.** *Given a demasking function f, a superstring S, and a binary mask M, such that $|M| = |S|$, we call a triplet $\mathcal{S} = (f, S, M)$ a* function-assigned masked superstring *or* f-masked superstrings*, and abbreviate it as* f-MS.
If $\exists Q \in \Sigma^k$ such that $f(\lambda(S, M, Q)) =$ `invalid`, we call the f-MS invalid.

Function f	Definition	Comprehensive	Use cases
or	1 if $\|\lambda\|_1 > 0$ 0 if $\|\lambda\|_1 = 0$	yes	• MS (Sec. 3) and (r)SPSS (App. A) • f for union (Sec. 4)
xor	1 if $\|\lambda\|_1$ is odd 0 if $\|\lambda\|_1$ is even	yes	• f for sym. difference (Sec. 4)
and	1 if $\lambda \neq \epsilon \wedge \|\lambda\|_0 = 0$ 0 if $\lambda = \epsilon \vee \|\lambda\|_0 > 0$	yes	• allowing ON occurrences for ghost k-mers (App. C)
$[a,b]$-**threshold** $(1 \leq a \leq b)$	1 if $a \leq \|\lambda\|_1 \leq b$ 0 otherwise	iff $a = 1$	• f_o for intersection (Sec. 4) • f_o for set difference (Sec. 4)
one-or-nothing	1 if $\|\lambda\|_1 = 1$ 0 if $\|\lambda\|_1 = 0$ `invalid` otherwise	yes	• f_1, f_2 for sym. difference (Sec. 4) • f_1, f_2 for intersection (Sec. 4) • f_1 for set difference (Sec. 4)
two-or-nothing	1 if $\|\lambda\|_1 = 2$ 0 if $\|\lambda\|_1 = 0$ `invalid` otherwise	no	• f_2 for set difference (Sec. 4)
all-or-nothing	1 if $\lambda \neq \epsilon \wedge \|\lambda\|_0 = 0$ 0 if $\|\lambda\|_1 = 0$ `invalid` otherwise	yes	• omitting mask rank in membership queries (App. C)

Table 1 Common demasking functions used throughout the paper. $\lambda(f, S, M)$ is abbreviated as λ. All the mentioned non-comprehensive functions still satisfy properties (P1,4) from Definition 4.

Now, for a valid f-MS, we can generalize Equation (2) for k-mer decoding as

$$K = \{Q \in \Sigma^k \mid f(\lambda(S, M, Q)) = 1\}. \tag{3}$$

As shown in [59], the most expensive part of MS computation (and thus also f-MS) is finding a short superstring. It is therefore natural to ask whether we require a superstring recomputation after performing a set operation. We now show that as long as function f in the f-MS representation is comprehensive, the superstring may remain the same and only a typically much simpler mask optimization may be needed.

▶ **Lemma 6** (encoding). *Let f be a comprehensive demasking function, K be a k-mer set, and S its arbitrary superstring. Then, there exists a mask M such that (f, S, M) is valid and represents K.*

Proof. The lemma is a direct consequence of property (P2) in Definition 4. ◀

Similarly, we can change the demasking functions f to another comprehensive function only by recoding the mask accordingly. For functions that are not comprehensive, such recording may be impossible.

▶ **Lemma 7** (recoding). *Let (f, S, M) be a valid f-MS representing a k-mer set K. Then, for every comprehensive demasking function f', there exists a valid mask M' such that (f', S, M') represents K.*

Proof. The existence of mask M' follows directly from Lemma 6 for the k-mer set K, superstring S, and function f'. ◀

▶ **Example 8.** Consider Example 1 with the set of 3-mers $K = \{\texttt{ACG}, \texttt{GGG}\}$ and the masked superstring `AcGGgg`. For the query k-mer $Q = \texttt{GGG}$, the occurrence function for is $\lambda(S, M, Q) = (1, 1)$. The result of demasking then $f(\lambda(S, M, Q))$ then depends on the specific function f: for **or** it evaluations to 1 and thus `GGG` is represented; however, for **xor** the result would be 0

and GGG would be a ghost k-mer, and thus not represented. As **xor** is comprehensive, we can recode the f-MS from **or** to **xor** in a way it would still represent K by changing the mask, for instance, to AcgGgg.

4 f-MS as an Algebraic Framework

In this section, we describe on the conceptual level how to perform set operations on k-mer sets by simply concatenating individual f-MS and choosing suitable demasking functions f. In Section 5, we deal with implementing this concept into data structures for k-mers, specifically, into the FMSI index [60].

4.1 Concatenation as an Elementary Low-Level Operation

We define concatenation on f-MS as concatenating the underlying superstrings and masks for all possible input and output functions f.

▶ **Definition 9.** *Given f-MS (f_1, S_1, M_1) and (f_2, S_2, M_2), we define (f_1, f_2, f_o)-concatenation as the operation taking these two f-MS and producing the result $(f_o, S_1 + S_2, M_1 + M_2)$. We denote this operation by $+_{f_1,f_2,f_o}$.*

Note that Definition 9 can be easily extended to more than two input f-MS. In the case that all the functions are the same, i.e. $f = f_1 = f_2 = f_o$, we call it *f-concatenation* or just *concatenation* if f is obvious from the context.

▶ **Definition 10.** *We call the set operations that can be performed with $f_1 = f_2 = f_o$* function-preserving set operations*. The operations that cannot be performed with a single function are called* function-transforming set operations*.*

Furthermore, note that while the set of appearing k-mers of $S_1 + S_2$ clearly contains the union of appearing k-mers of S_1 and of S_2, additional new occurrences of k-mers may appear at the boundary of the two superstrings. These newly appearing k-mers may not be appearing in any of the superstrings S_1 and S_2, and we refer to them as *boundary k-mers*. The occurrences of appearing k-mers of $S_1 + S_2$ that overlap both input superstrings are called *boundary occurrences*.

4.2 Function-Preserving Set Operations

Union. As implicitly shown in [59], concatenating MS, which are **or**-MS in our notation, acts as union on the represented sets, that is, the resulting represented set is the union of the original represented sets. This allows **or**-MS to generalize (r)SPSS representations, since any set of k-mers in the (r)SPSS representation can be directly viewed as an **or**-MS by concatenating the individual simplitigs/matchtigs.

We show that **or** is the *only* comprehensive demasking function that acts as union on the represented sets; see Appendix A for details. We further demonstrate this uniqueness even on the level of matchtigs and therefore, **or**-MS are the only f-MS that generalize (r)SPSS representations.

Symmetric difference. Next, we observe that **xor** naturally acts as the symmetric difference set operation, i.e., concatenating two **xor**-MS results in a **xor**-MS representing the symmetric difference of the original sets. Indeed, recall that using **xor** implies that a k-mer is represented if and only if there is a odd number of ON occurrences of that k-mer.

Observe that the boundary occurrences of k-mers do not affect the resulting represented set as those have zeros in the mask. Thus, if a k-mer is present in both sets, it has an even number of ON occurrences in total and hence, is not represented in the result. Likewise, if a k-mer belongs to exactly one input set, it has an odd number of ON occurrences in this input set and an even number (possibly zero) in the other; thus, it is represented in the result. As any appearing k-mer is either boundary or appears in one of the MS, the result corresponds to the symmetric difference.

4.3 Function-Transforming Set Operations

Intersection. After seeing functions for union and symmetric difference operations, it might seem natural that there should be a function for intersection. This is however not the case as there exists no comprehensive demasking function acting as intersection, which we show in Appendix B.

We can circumvent the non-existence of a single demasking function acting as intersection by using possibly non-comprehensive demasking functions that are different for the result than for the input. We further show that such schemes have other applications beyond intersection.

To this end, we will need two different types of demasking functions:

- $[a,b]$-**threshold** function (where $0 < a \leq b$) is a demasking function that returns 1 whenever it receives an input of at least a ones and at most b ones and 0 otherwise. Note that unless $a = 1$, $[a,b]$-**threshold** functions are not comprehensive as they do not satisfy properties (P2) and (P3). The corresponding f-MS are denoted $[a,b]$-**threshold**-MS.
- The **one-or-nothing** function is a demasking function that returns 1 if there is exactly one 1 in the input, 0 if there are no 1s, and `invalid` if there is more than a single ON occurrence of the k-mer. Note that this function is comprehensive.

We now use these functions to perform any symmetric set operation on any number of input k-mer sets. Given N sets of k-mers, we compute a **one-or-nothing**-MS for each. This is always possible since **one-or-nothing** is a comprehensive demasking function and can be done by directly using the superstrings and masks computed by KmerCamel [59].

We then concatenate the individual **one-or-nothing**-MS. The result is not a valid **one-or-nothing**-MS in general, but it has the special property that each k-mer has as many ON occurrences as the number of sets in which it appears. We can therefore change the demasking function of the resulting f-MS from **one-or-nothing** to an $[a,b]$-**threshold** function. This will result in an $[a,b]$-**threshold**-MS that is always valid and the represented set will be exactly the k-mers that appear in at least a sets and at most b sets. Important $[a,b]$-**threshold**-MS in this setting include the following:

- The $[N,N]$-**threshold**-MS corresponds to taking the intersection of the represented sets.
- The $[1,N]$-**threshold**-MS is the **or**-MS and corresponds to taking the union.
- The $[1,1]$-**threshold**-MS corresponds to taking those k-mers that appear in exactly one of the original sets. In case of $N = 2$, this corresponds to the symmetric difference.

It is important to emphasize that we can use different $[a,b]$-**threshold** functions to alter the resulting k-mer set without changing the superstring or the mask. For instance, we can use the same superstring and mask to consider intersection and union simply by changing the function from $[N,N]$-**threshold** to $[1,N]$-**threshold**.

Arbitrary symmetric set operations. The same scheme, with more general demasking functions, can be used to implement *any symmetric set operation* ***op*** *on any number of sets.*

Indeed, given N, we again concatenate their **one-or-nothing**-MS in an arbitrary order. The symmetry of **op** implies that there is a set $S_N \subseteq \{0, 1, \ldots, N\}$ such that a k-mer belongs to the set resulting from applying **op** if and only if it is in a input sets for some $a \in S_N$. The sets S_N for $N = 1, 2, \ldots$ can be directly transferred into a demasking function $f_{\mathbf{op}}$ that models **op**; however, $f_{\mathbf{op}}$ may not satisfy the property (P4) from Definition 4, i.e., that we can compute it in $O(1)$ time.

Set difference. Having seen how to perform symmetric set operations, we deal with asymmetric ones, focusing on the set difference of k-mer sets $A \setminus B$. Clearly, we cannot use the same demasking function f to represent both A and B as it would be impossible to distinguish the sets after concatenation. Hence, we use different functions to represent A and B, namely,

- represent A using a [1,1]-**threshold**-MS,
- represent B using a [2,2]-**threshold**-MS, and
- interpret the result as a [1,1]-**threshold**-MS.

This computes the difference correctly as all k-mers represented in B are treated as ghosts in the result, the k-mers from A but not from B still have a single ON occurrence and thus are correctly considered represented, and finally, the ghost k-mers in either of the initial sets or the boundary k-mers have no influence on the result. The same functions can be used if we subtract more than a single set. Furthermore, this scheme can be generalized to *any set operations on any number of sets*, by representing the i-th input set with [i,i]-**threshold**-MS and using a suitable demasking function for the result of the concatenation (constructed similarly as $f_{\mathbf{op}}$ for symmetric operation **op** above).

The downside to this approach is that the [2,2]-**threshold** function is not comprehensive and we cannot simply use any superstring of k-mers in B, but we need a superstring such that every k-mer of B appears at least twice, which can for instance be achieved by doubling the computed superstring of B. We remark that this is the best we can do as set difference cannot be achieved with comprehensive functions solely as we show in Appendix B.

Other applications. Furthermore, there are many more demasking functions that can be used with f-MS, although they may not correspond to set operations; we mention the **and** and **all-or-nothing** demasking functions in Appendix C (see also Table 1).

5 Indexing f-MS with the FMSI index

To support set operations on k-mer sets, while allowing fast k-mer queries, we utilize the FMSI index [60] introduced in our concurrent work. We first give an overview of the FMSI index [60] and then describe how to generalize the query algorithm in FMSI to support arbitrary demasking functions f. Next, we show that performing set operations narrows down to merging the Burrows-Wheeler Transform and provide algorithms for changing the demasking function f. Last, if after several set operations the size of the index gets too large, one can improve space efficiency by a *compaction*, i.e., recomputing the representation.

FMSI index. The FMSI index [60] for a k-mer set K is constructed from its masked superstring (M, S) maximizing the number of ones, which can also be seen as **all-or-nothing**-masked superstring (Appendix C). The FMSI index consists of the Burrows-Wheeler transform (BWT) [16] of S with an associated rank data structure [31], and the *SA-transformed mask* M' which is a bit-vector of the same length as S, where $M'[i] = M[j_i - 1 \mod |M|]$ where j_i is the starting position of the lexicographically i-th suffix of S. Optionally, FMSI index can also construct the *kLCP array* [54], where $kLCP[i] = 1$ if the i-th and $(i+1)$-th suffix

share a prefix of $k-1$ characters. FMSI can be constructed in linear time through Masked BWT [60], a tailored variant of the classical BWT [16]. FMSI can index a k-mer Q in $O(k)$ time by first computing the range of occurrences of Q in the suffix-array coordinates. Then, since the bits of the SA-transformed mask in this range correspond to the mask symbols of occurrences of Q and since the mask has the maximum number of ones, the presence or absence of Q in the represented k-mer set can be determined from any of those bits [60]. If kLCP is used, streaming queries can be answered in $O(1)$ time per k-mer [60]. Here, we only consider the FMSI index without kLCP, which requires $2 + o(1)$ bits of memory per distinct k-mer under the spectrum-like property and at most $3 + o(1)$ bits per superstring character in the general case [60]. In addition, we consider the rank data structure also for the SA-transformed mask, which does not asymptotically increase complexity, i.e., costs only another $o(1)$ bits per superstring character [31].

Efficient queries with arbitrary demasking functions. In [60, Algorithm 2] we describe the $O(k)$-time query algorithm for **or**-MS with masks having the maximum number of ones, which can be viewed as **all-or-nothing**-MS (Appendix C). In Lemma 11 we generalize the result of [60, Lemma 4] to f-MS with comprehensive f, which can be directly translated to an algorithm for querying general f-MS with the same time guarantees.

▶ **Lemma 11.** *Consider a query for k-mer Q on an f-MS (f, S, M) representing a k-mer set K such that f is comprehensive. Let M' be the corresponding SA-tranformed mask [60] and let $[i, j)$ be the range of sorted rotations of S starting with a k-mer Q. Then the presence or absence of Q in K can be determined in $O(1)$ time.*

Proof. From [60, Lemma 1], $M'[x]$ for $x \in [i, j)$ corresponds to the mask symbol of a particular occurrence of Q. Therefore $|\lambda(S, M, Q)|_1 = \mathrm{rank}_1(M', j) - \mathrm{rank}_1(M', i)$, which can be computed in $O(1)$ time using two rank queries on the mask; here, $\mathrm{rank}_1(M', i) = \sum_{a=0}^{i-1} M'[a]$ is the number of ones on coordinates $0, \ldots, i-1$ in M', computed by the rank data structure. Furthermore, $|\lambda(S, M, Q)|_0 = |\lambda(S, M, Q)| - |\lambda(S, M, Q)|_1 = j - i - |\lambda(S, M, Q)|_1$. Since f is comprehensive and in particular, commutative, $f(\lambda(S, M, Q))$ can be evaluated from the two quantities in constant time. ◀

To query a k-mer we can then simply use backwards search to get the range of occurrences of Q and then apply Lemma 11. The same adjustment works also for positive streaming queries which only require $O(1)$ time per query k-mer [60].

Set operations in linear time in the index sizes. In Section 4, we describe how to implement set operations with f-MS via masked superstring concatenation. Performing the operation on indexes boils down to merging two BWTs using any algorithm for BWT merging, for example [30] which runs in linear time. To merge the SA-tranformed masks, we attach the mask symbols to the corresponding characters of BWT, as described in [60]; hence, the existing algorithms for BWT merging can be adjusted in a straightforward way to merge the SA-tranformed masks.

***f*-MS mask recasting for f transformation.** To change the demasking function f to a different one without altering the represented k-mer set and the underlying superstring, we may need to *recast* the mask. Although the recasting procedure depends on the specific function f used, for all comprehensive functions mentioned in Table 1, recasting can be done either by maximizing the number of 1s in the mask (**and** and **all-or-nothing**), or by minimizing the number of 1s (all other functions in Table 1). If an f-MS is represented in the original string form, this can be achieved in linear time using a single/two-pass algorithms, respectively, as described in [59]. For f-MS in the suffix-array coordinates, we can export the f-MS, then recast the mask, and index the result.

***f*-MS compaction.** If an f-MS contains too many redundant copies of individual k-mers, e.g., if an f-MS is obtained by concatenating multiple input f-MS, it might be desirable to *compact* it, i.e., reoptimize its support superstring. This can be performed in time $O(k|K|)$, by exporting the f-MS, counting the number of ON and OFF occurrences of each k-mer, constructing the represented k-mer set, and computing its **or**-MS [59]. We leave it to future work to design algorithms for compaction and for mask recasting directly in the FMSI index without the need to export the f-MS.

6 Implementation and a Proof-of-Concept Experiment

Implementation in the FMSI tool. We implemented this functionality as a proof-of-concept in the FMSI program [60], which is available under the MIT license on GitHub (`https://github.com/OndrejSladky/fmsi`) and distributed via BioConda. Merging of several indexes is currently implemented only by exporting individual superstrings, concatenating them as masked superstrings, and reindexing the result. Querying is a slight modification of the original implementation as described in [60], according to Lemma 11. We emphasize the our implementation is prototype, to demonstrate feasibility of performing set operations without the necessity to recompute a representation, and can be substantially optimized.

Proof-of-concept experiment. We conducted a simple experiment with genomes of *C. elegans* (`NC_003279.8`, 100M base pairs) and *C. briggsae* (`NC_013489.2`, 108M base pairs), performing the union, intersection, and symmetric difference operations on their sets of k-mers. Our proposed pipeline for these operations, as depicted in Figure 1, consists of five steps: First, we compute a textual representation of the k-mer sets interpreted as **or**-MS. In our experiments, this was done using KmerCamel's global greedy algorithm [59]. Second, we recast the mask to the desired demasking function, specifically we keep **or** for union and change to **one-or-nothing** for intersection and to **xor** for symmetric difference. In the case of **or**-MS computed by KmerCamel, mask recasting is actually not needed as the output already minimizes the number of 1s in the mask. Then we index the f-MS using FMSI [60] (this can be done even before mask recasting). The last two steps are concatenating the two indexes by index merging and compacting the resulting index if needed. Note that once indexed, we can ask membership queries on the resulting k-mer sets.

We evaluated the superstring length of each computed f-MS, and the memory requirements to perform queries on the indexed individual and concatenated f-MS, both before and after compaction. The results for intersection of the two roundworm genomes for $k = 21$ are depicted in Figure 1. On the k-mer sets of both source genomes, which satisfy the spectrum-like property (SLP), the indexed f-MS required around 2.7 bits per distinct k-mer to perform queries. After taking the intersection, however, only about 1% of distinct k-mers remained, and the resulting k-mer set is far from SLP as the new MS representation after compaction requires three times more characters than the number of distinct k-mers. The memory requirements of FMSI per distinct k-mer are high as the resulting set has relatively few k-mers, less than one million, so latent memory to run FMSI was relatively significant. Note also that the fact that compaction significantly reduces the MS length highly depends on the particular use case, namely on the proportion of represented k-mers in the result. For union and symmetric difference for the same data, the compaction lead to negligible length reduction.

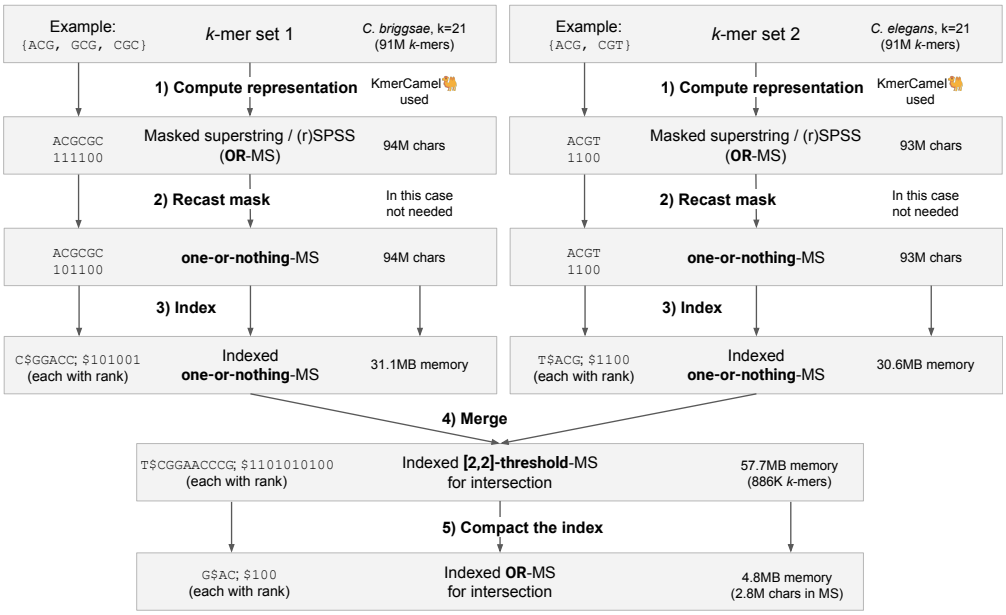

Figure 1 Illustration of the k-mer set operations workflow using f-masked superstrings: an example of intersection of two sets using FMSI [60]. The workflow also contains an illustrative example on a set of 3-mers as well as experimental results on *C. briggsae* and *C. elegans* genomes with $k = 21$; namely, we show the number of masked superstring characters after each change, the number of distinct k-mers in each set, and for each indexed representation, the memory required to perform queries on the underlying set.

In contract, CBL [46][1] required about 950 MB of memory for processing queries for both original genomes, which translates to about 83 bits per distinct k-mer. After computing the index for the intersection (from the indexes of the source genomes), the memory dropped to 46.5 MB, still significantly larger than the memory of FMSI.

For results with symmetric difference, union, and other values of k, see the supplementary repository (`https://github.com/OndrejSladky/f-masked-superstrings-supplement`).

7 Conclusion and Outlook

We have proposed f-masked superstrings (f-MS) as an algebraic data type for k-mer sets that allows for seamless execution of set operations. It is primarily based on equipping masked superstrings (MS) from [59] with a demasking function f, and we have thoroughly investigated several natural demasking functions, demonstrating that set operations on k-mer sets can be carried out simply by masked superstring concatenation or, if indexed, by merging their masked Burrows-Wheeler transform from [60]. This leads to a simple data structure that simultaneously allows for beyond worst-case compressibility, answering exact membership queries, and efficiently performing set operations on the k-mer sets, without the

[1] Obtained from `https://github.com/imartayan/CBL`, version at commit '`328bcc6`'. The index was computed on canonical k-mers, to handle reverse complements, that is, we ran '`cbl build -c`'. CBL was compiled using '`RUSTFLAGS="-C target-cpu=native" K={kmer-size} PREFIX_BITS=28 \\ cargo +nightly build ---release ---examples ---target-dir target.k_{kmer-size}`'.

costly operation of recomputing the underlying representation. Another major advantage is the versality of our concept as it can in fact be combined with (repetitive) Spectrum Preserving String Sets [12, 51, 58] instead of (more general) masked superstrings.

The main practical limitation of our work is the current implementation of the index merging, which is very slow and not using the state-of-the-art algorithms for BWT merging [30]. Furthermore, our proof-of-concept experiment is only meant to demonstrate feasibility, and we leave a more thorough evaluation, using various datasets and including a comparison to other tools for set operations, such as CBL [46], to future work.

Our work opens up several research directions for future theoretical investigation. Currently, to deal with non-comprehensive demasking functions, which are necessary for asymmetric set operations, multiple copies of input masked superstrings are needed. This leads to a natural question whether this framework can be further generalized, possibly by extending the mask alphabet, to allow for more efficient asymmetric set operations. On the algorithmic level, our work relies on efficient algorithms for merging the Burrows-Wheeler transform, as well as merging the supporting data structures such as the kLCP array [54]. Moreover, it is open how to directly perform certain operations with f-masked superstrings indexed with the masked Burrows-Wheeler transform [60], namely, we seek algorithms for mask recasting and index compaction that do not necessitate to export the masked superstring, but rather work locally with the BWT of the superstring and the SA-transformed mask. Furthermore, as our work deals only with set operations, we open the question of performing single insertions and deletions in a more efficient way than performing these through set operations; note that for comprehensive demasking functions, deletions in the representation can be handled efficiently by changing the corresponding mask bits.

In conclusion, while the primary contributions of this paper are conceptual, they pave the path towards a space- and time-efficient library for k-mer sets that would include all of these features. In the light of advances in efficient superstring approximation algorithms and BWT-based indexing, we believe that the f-masked superstring framework is a useful step towards designing appropriate data structures for this library, which is now mainly an engineering challenge.

References

1. Jarno Alanko, Bahar Alipanahi, Jonathen Settle, Christina Boucher, and Travis Gagie. Buffering updates enables efficient dynamic de bruijn graphs. *Computational and structural biotechnology journal*, 19:4067–4078, 2021.
2. Jarno N Alanko, Simon J Puglisi, and Jaakko Vuohtoniemi. Small searchable κ-spectra via subset rank queries on the spectral Burrows-Wheeler transform. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA23)*, pages 225–236. SIAM, 2023. doi:10.1137/1.9781611977714.20.
3. Jarno N Alanko, Jaakko Vuohtoniemi, Tommi Mäklin, and Simon J Puglisi. Themisto: a scalable colored k-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes. *Bioinformatics*, 39(Supplement_1):i260–i269, 2023. doi:10.1093/bioinformatics/btad233.
4. Bahar Alipanahi, Alan Kuhnle, Simon J Puglisi, Leena Salmela, and Christina Boucher. Succinct dynamic de bruijn graphs. *Bioinformatics*, 37(14):1946–1952, 2021.
5. Fatemeh Almodaresi, Jamshed Khan, Sergey Madaminov, Michael Ferdman, Rob Johnson, Prashant Pandey, and Rob Patro. An incrementally updatable and scalable system

for large-scale sequence search using the Bentley-Saxe transformation. *Bioinformatics*, 38(12):3155–3163, 2022. doi:10.1093/bioinformatics/btac142.

6. Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. COBS: a compact bit-sliced signature index. In *String Processing and Information Retrieval: 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7–9, 2019, Proceedings 26*, pages 285–303. Springer, 2019. doi:10.1007/978-3-030-32686-9_21.
7. Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In Benjamin J. Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7534 of *Lecture Notes in Computer Science*, pages 225–235. Springer, 2012. doi:10.1007/978-3-642-33122-0_18.
8. Phelim Bradley, Henk C Den Bakker, Eduardo PC Rocha, Gil McVean, and Zamin Iqbal. Ultrafast search of all deposited bacterial and viral genomic data. *Nature Biotechnology*, 37(2):152–159, 2019.
9. Phelim Bradley, N Claire Gordon, Timothy M Walker, Laura Dunn, Simon Heys, Bill Huang, Sarah Earle, Louise J Pankhurst, Luke Anson, Mariateresa De Cesare, et al. Rapid antibiotic-resistance predictions from genome sequence data for staphylococcus aureus and mycobacterium tuberculosis. *Nature Communications*, 6(1):10063, 2015. doi:10.1038/ncomms10063.
10. Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016. doi:10.1038/nbt.3519.
11. Karel Břinda. Novel computational techniques for mapping and classification of Next-Generation Sequencing data. PhD thesis, Université Paris-Est, 2016. doi:10.5281/zenodo.1045317.
12. Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *Genome Biology*, 22(96), 2021. doi:10.1186/s13059-021-02297-z.
13. Karel Břinda, Alanna Callendrello, Kevin C Ma, Derek R MacFadden, Themoula Charalampous, Robyn S Lee, Lauren Cowley, Crista B Wadsworth, Yonatan H Grad, Gregory Kucherov, et al. Rapid inference of antibiotic resistance and susceptibility by genomic neighbour typing. *Nature Microbiology*, 5(3):455–464, 2020. doi:10.1038/s41564-019-0656-6.
14. Karel Břinda, Leandro Lima, Simone Pignotti, Natalia Quinones-Olvera, Kamil Salikhov, Rayan Chikhi, Gregory Kucherov, Zamin Iqbal, and Michael Baym. Efficient and robust search of microbial genomes via phylogenetic compression. *bioRxiv*, 2023.04.15.536996, 2023. doi:10.1101/2023.04.15.536996.
15. Karel Břinda, Kamil Salikhov, Simone Pignotti, and Gregory Kucherov. Prophyle 0.3.1.0. *Zenodo*, 5281, 2017. URL: https://prophyle.github.io, doi:10.5281/zenodo.5237391.
16. Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
17. Rayan Chikhi. K-mer data structures in sequence bioinformatics. HDR thesis, Institut Pasteur Ecole Doctorale “EDITE”, 2021. URL: http://rayan.chikhi.name/pdf/RChikhi-HDR.pdf.
18. Rayan Chikhi, Jan Holub, and Paul Medvedev. Data structures to represent a set of k-long DNA sequences. *ACM Computing Surveys*, 54(1):17:1–17:22, 2022. doi:10.1145/3445967.

19. Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T. Simpson, and Paul Medvedev. On the representation of de Bruijn graphs. In Roded Sharan, editor, *Research in Computational Molecular Biology*, pages 35–55, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-05269-4_4.
20. Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016. doi:10.1093/bioinformatics/btw279.
21. Thomas C. Conway and Andrew J. Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–486, 2011. doi:10.1093/bioinformatics/btq697.
22. Victoria G. Crawford, Alan Kuhnle, Christina Boucher, Rayan Chikhi, and Travis Gagie. Practical dynamic de Bruijn graphs. *Bioinformatics*, 34(24):4189–4195, 2018. doi:10.1093/bioinformatics/bty500.
23. Sebastian Deorowicz, Agnieszka Debudaj-Grabysz, and Szymon Grabowski. Disk-based k-mer counting on a pc. *BMC bioinformatics*, 14:1–12, 2013.
24. Jason Fan, Jamshed Khan, Giulio Ermanno Pibiri, and Rob Patro. Spectrum preserving tilings enable sparse and modular reference indexing. In Haixu Tang, editor, *Research in Computational Molecular Biology - 27th Annual International Conference, RECOMB 2023, Istanbul, Turkey, April 16-19, 2023, Proceedings*, volume 13976 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2023. doi:10.1007/978-3-031-29119-7_2.
25. Jason Fan, Jamshed Khan, Noor Pratap Singh, Giulio Ermanno Pibiri, and Rob Patro. Fulgor: a fast and compact k-mer index for large-scale matching and color queries. *Algorithms for Molecular Biology*, 19(1):3, 2024. doi:10.1186/S13015-024-00251-9.
26. Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, SFCS-00. IEEE Comput. Soc, 2000. doi:10.1109/sfcs.2000.892127.
27. Gaurav Gupta, Minghao Yan, Benjamin Coleman, Bryce Kille, R. A. Leo Elworth, Tharun Medini, Todd Treangen, and Anshumali Shrivastava. Fast processing and querying of 170TB of genomics data via a Repeated And Merged BloOm filter (RAMBO). In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2226–2234, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3448016.3457333.
28. Khodor Hannoush, Camille Marchet, and Pierre Peterlongo. Cdbgtricks: Strategies to update a compacted de bruijn graph. *bioRxiv*, 2024. URL: https://www.biorxiv.org/content/early/2024/05/28/2024.05.24.595676, arXiv:https://www.biorxiv.org/content/early/2024/05/28/2024.05.24.595676.full.pdf, doi:10.1101/2024.05.24.595676.
29. Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. *Genome Biology*, 21(1):1–20, 2020. doi:10.1186/s13059-020-02135-8.
30. James Holt and Leonard McMillan. Merging of multi-string bwts with applications. *Bioinformatics*, 30(24):3524–3531, August 2014. doi:10.1093/bioinformatics/btu584.
31. Guy Joseph Jacobson. *Succinct static data structures*. PhD thesis, Carnegie Mellon University, USA, 1988. AAI8918056.
32. Lauris Kaplinski, Maarja Lepamets, and Maido Remm. GenomeTester4: a toolkit for performing basic set operations - union, intersection and complement on k-mer lists. *GigaScience*, 4(1):s13742–015–0097–y, 12 2015. doi:10.1186/s13742-015-0097-y.

33. Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Marc Zimmermann, Christopher Barber, Gunnar Rätsch, and André Kahles. Indexing all life's known biological sequences. *bioRxiv*, 2020.10.01.322164, 2024. doi:10.1101/2020.10.01.322164.
34. Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 05 2017. doi:10.1093/bioinformatics/btx304.
35. Téo Lemane, Paul Medvedev, Rayan Chikhi, and Pierre Peterlongo. kmtricks: efficient and flexible construction of Bloom filters for large sequencing data collections. *Bioinformatics Advances*, 2(1):vbac029, 04 2022. doi:10.1093/bioadv/vbac029.
36. Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, 2012. doi:10.1093/bioinformatics/bts280.
37. Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013. doi:10.48550/arXiv.1303.3997.
38. Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.
39. Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010. doi:10.1093/bioinformatics/btp698.
40. Antoine Limasset, Guillaume Rizk, Rayan Chikhi, and Pierre Peterlongo. Fast and scalable minimal perfect hashing for massive key sets. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, volume 75 of *LIPIcs*, pages 25:1–25:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/lipics.sea.2017.25.
41. Abdullah-Al Mamun, Soumitra Pal, and Sanguthevar Rajasekaran. Kcmbt: ak-mer counter based on multiple burst trees. *Bioinformatics*, 32(18):2783–2790, 2016.
42. Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011.
43. Camille Marchet, Christina Boucher, Simon J Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, 2021. doi:10.1101/gr.260604.119.
44. Camille Marchet, Zamin Iqbal, Daniel Gautheret, Mikaël Salson, and Rayan Chikhi. REINDEER: efficient indexing of k-mer presence and abundance in sequencing datasets. *Bioinformatics*, 36(Supplement-1):i177–i185, 2020. doi:10.1093/bioinformatics/btaa487.
45. Camille Marchet, Maël Kerbiriou, and Antoine Limasset. Blight: efficient exact associative structure for k-mers. *Bioinformatics*, 37(18):2858–2865, 2021. doi:10.1093/bioinformatics/btab217.
46. Igor Martayan, Bastien Cazaux, Antoine Limasset, and Camille Marchet. Conway–bromage–lyndon (cbl): an exact, dynamic representation of k-mer sets. *Bioinformatics*, 40(Supplement_1):i48–i57, 06 2024. arXiv:https://academic.oup.com/bioinformatics/article-pdf/40/Supplement_1/i48/58354678/btae217.pdf, doi:10.1093/bioinformatics/btae217.
47. Martin D. Muggli, Bahar Alipanahi, and Christina Boucher. Building large updatable colored de Bruijn graphs via merging. *Bioinformatics*, 35(14):i51–i60, 2019. doi:10.1093/bioinformatics/btz350.

48. Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, 2017. doi:10.1038/nmeth.4197.
49. Giulio Ermanno Pibiri. Sparse and skew hashing of K-mers. *Bioinformatics*, 38(Supplement_1):i185–i194, 2022. doi:10.1093/bioinformatics/btac245.
50. Amatur Rahman. Compression algorithms for de Bruijn graphs and uncovering hidden assembly artifacts. PhD thesis, The Pennsylvania State University, 2023.
51. Amatur Rahman and Paul Medevedev. Representation of k-mer sets using spectrum-preserving string sets. *Journal of Computational Biology*, 28(4):381–394, 2021. PMID: 33290137. doi:10.1089/cmb.2020.0431.
52. Arang Rhie, Brian P Walenz, Sergey Koren, and Adam M Phillippy. Merqury: reference-free quality, completeness, and phasing assessment for genome assemblies. *Genome biology*, 21:1–27, 2020.
53. Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. Dsk: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, 2013.
54. Kamil Salikhov. Efficient algorithms and data structures for indexing dna sequence data. PhD thesis, Université Paris-Est, 2017.
55. Kamil Salikhov, Karel Břinda, Simone Pignotti, and Gregory Kucherov. ProPhex. https://github.com/prophyle/prophex, 2018. doi:10.5281/zenodo.1247432.
56. Sebastian Schmidt. Unitigs are not enough: the advantages of superunitig-based algorithms in bioinformatics. PhD thesis, University of Helsinki, 2023.
57. Sebastian Schmidt and Jarno N. Alanko. Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time. *Algorithms for Molecular Biology*, 18(1):5, 2023. doi:10.1186/s13015-023-00227-1.
58. Sebastian Schmidt, Shahbaz Khan, Jarno N. Alanko, Giulio E. Pibiri, and Alexandru I. Tomescu. Matchtigs: minimum plain text representation of k-mer sets. *Genome Biology*, 24(1):136, 2023. doi:10.1186/s13059-023-02968-z.
59. Ondřej Sladký, Pavel Veselý, and Karel Břinda. Masked superstrings as a unified framework for textual k-mer set representations. *bioRxiv*, 2023.02.01.526717, 2023. doi:10.1101/2023.02.01.526717.
60. Ondřej Sladký, Pavel Veselý, and Karel Břinda. FroM Superstring to Indexing: a space-efficient index for unconstrainedk-mer sets using the masked burrows-wheeler transform (MBWT). *bioRxiv*, November 2024. URL: http://dx.doi.org/10.1101/2024.10.30.621029, doi:10.1101/2024.10.30.621029.
61. 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 Biology*, 13(7):e1002195, 2015. doi:10.1371/journal.pbio.1002195.
62. Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):1–12, 2014. doi:10.1186/gb-2014-15-3-r46.

A Proof of Uniqueness of Union Function

Masked superstrings [59], which we call **or**-MS, have the important property that concatenating them results in the union of represented k-mers. This property makes it possible for MS

to generalize (r)SPSS representations [59] as unifying individual simplitigs/matchtigs results in correctly representing the union of respective represented k-mer sets. In this section, we show that **or**-MS are the only f-MS with any of these properties (acting as union and generalizing (r)SPSS).

▶ **Theorem 12.** ***or*** *is the only comprehensive demasking function f such that for any two k-mer sets K and K' and any of their valid f-MS (f, S, M) and (f, S', M'), respectively, their concatenation $(f, S + S', M + M')$ is a valid f-MS representing the set $K \cup K'$.*

Proof. For a contradiction assume there is a comprehensive demasking function f different than **or** that satisfies the above. Consider the smallest n such that f behaves differently than **or** for a length-n input, meaning that there is $x \in \{0,1\}^n$ not equal to the all-zeros vector such that $f(x) \neq 1$. As f is comprehensive, it cannot happen that $f(x) = 1$ for all $x \in \{0,1\}^n$ by Definition 4, and moreover, $f((1)) = 1$, implying $n > 1$. Fix a k-mer, for simplicity $\mathtt{A}^k$ (although similar approach works for all k-mers). We take the first f-MS to be the k-mer with mask being $M_0 = x_0$ and $M_i = 0$ for the remaining $k-1$ positions $i > 0$. The second f-MS is $\mathtt{CA}\ldots\mathtt{A}$ where $\mathtt{A}$ appears $n + k - 2$ times with the mask being: $M_0 = 0$, $M_i = x_i$ for $i = 1, \ldots, k$, and $M_i = 0$ for $i > k$. At least one of the represented sets contains the k-mer as $x \neq 0$ and n is the but the resulting f-MS is either invalid or does not contain the k-mer in the represented set as $f(x) \neq 1$, a contradiction. ◀

▶ **Theorem 13.** ***or*** *is the only comprehensive demasking function f such that for any sequence of f-MS, where individual superstrings are matchtigs, the concatenation of all the f-MS represents the union of represented k-mers.*

Proof. It is sufficient to find a construction of matchtigs such that we can construct an arbitrary sequence of ones and zeros at the occurrences of a given k-mer and the rest follows similarly as in Theorem 12.

We do this with k-mer $\mathtt{CG}$ and matchtigs $\mathtt{Cg}$ and $\mathtt{Gc}$. Consider the counterexample sequence of occurring ones and zeros from Theorem 12. For every one in the sequence, we add the matchtig $\mathtt{Cg}$ and for each m consecutive zeros, we add $m + 1$ times the matchtig $\mathtt{Gc}$, since at the boundary of two $\mathtt{Gc}$ matchtigs an OFF occurrence of k-mer $\mathtt{CG}$ appears. At any other boundary, the k-mer $\mathtt{CG}$ does not appear, therefore the construction is correct. The rest of the proof follows a similar argument as in Theorem 12. ◀

Note, however, that the same does not hold if we want to represent simplitigs/SPSS solely. As an individual k-mer cannot appear more than once with an ON occurrence, any comprehensive function generalizes SPSS representations if it satisfies that if there is one ON occurrence of a k-mer, it returns 1, and if there is none, it returns 0.

B Limits of Performing Set Operations using Comprehensive Functions

In this section, we prove theoretical limitations of performing set operations by concatenation of f-MS with comprehensive demasking functions. First, we show that intersection cannot be a function-preserving set operation and second, we prove that it is impossible to use only comprehensive demasking functions for input sets for the set difference operation. In Section 4, we show how to overcome these limits via careful choice of demasking functions that are not comprehensive.

B.1 Non-Existence of a Preserved Comprehensive Function for Intersection.

We show that there is no comprehensive demasking function that acts as the intersection when concatenating f-MS. In a nutshell, this impossibility is caused by the fact that if there is a k-mer Q that occurs exactly once in the input MS with `1` in the mask, then after concatenation, it will still occur once with `1` in the mask, so under any comprehensive f the k-mer would appear as if it was in the intersection.

▶ **Theorem 14.** *There is no comprehensive demasking function f with the property that the result of f-concatenation of two f-MS always represents the intersection of the originally represented k-mer sets.*

Proof. Let f be any comprehensive demasking function. Consider MS `A` and `C`, each representing a single 1-mer. Their concatenation is `AC`. Since $f((1)) = 1$ by the comprehensiveness of f, the concatenation represents both 1-mer `A` and `C`. However, the intersection is empty and thus, f cannot be used to compute the intersection from the concatenation. ◀

Note that the proof cannot be generally extended to all demasking function as there exist non-comprehensive demasking functions acting as the intersection on the represented sets upon concatenation, for instance the constant zero function. However, since the constant zero function always represents the empty set, it is of no use in practice.

We further remark that although we have for convenience used the property (P3) from the definition of comprehensive functions, the proof in fact relies only on the property (P2) and holds even if we consider not only 1-mers, that is, a similar example can be constructed for any k.

B.2 Non-Existence of Comprehensive Input Functions for Set Difference.

We show that it is impossible to perform set difference of k-mer sets using f-MS with comprehensive functions solely.

▶ **Theorem 15.** *There is no demasking function f_o and no comprehensive demasking functions f_1 and f_2, such that the result of (f_1, f_2, f_o)-concatenation would always represent the set difference of the originally represented k-mer sets.*

Proof. Consider two masked superstrings and a k-mer Q which appears only once in each of the superstring. As an example, takse the set of 2-mers $K_1 = \{\texttt{CA}, \texttt{CG}\}$ and $K_2 = \{\texttt{AC}\}$ and their (shortest) superstrings $S_1 = \texttt{CACG}$ and $S_2 = \texttt{AC}$. Since the k-mer Q appears only once, from comprehesiveness, it must be represented as ON if it is present and as OFF otherwise, resulting that it is represented only based on its presence, the same in both MS. By symmetry of the output function, the result is the same if we concatenate $\mathcal{S}_1$ and $\mathcal{S}_2$ or the other way around. But it is not true in general that $K_1 - K_2 = K_2 - K_1$ (consider the example above). This gives a contradiction and concludes the proof. ◀

C Alternative Demasking Functions

In this section, we provide two other demasking functions that can be useful for some applications.

C.1 The all-or-nothing-masked superstrings

Perhaps the simplest approach to representing a set of k-mers is to mark all occurrences of represented k-mers with one, all ghost k-mers with zero, and treat all other masks as invalid. This corresponds to a function that returns 1 if it receives a list of ones, 0 if a list of zeros (or an empty list), and `invalid` otherwise. Alternatively, **all-or-nothing**-MS can be viewed as **or**-MS that maximize the number of ones in the mask.

This representation has been use in the FMSI index [60] as it allows to determine the presence of a k-mer from the mask symbol at an arbitrary occurrence of a particular k-mer, which enables the FMSI index to infer k-mer presence without any rank queries on the mask [60].

As another possible use case of **all-or-nothing**-MS, we could potentially achieve higher compressibility of the mask by realizing that we can infer the presence or absence of a k-mer from its first occurrence, which comes from the fact that a mask for a given set is unique. Thus, we can omit all symbols in the mask corresponding to any further occurrences of the k-mer, making the mask shorter and easier to store, while it can be easily reconstructed afterwards.

C.2 The and-MS

We could easily replace the **or** function with **and**. That is, we could consider a k-mer present if it is marked as present at *all* its occurrences, with the small difference that we consider a k-mer not represented if it does not appear, i.e., we consider the **and** of an empty binary string equal to 0, unlike in typical definitions of **and**. This ensures that the **and** function is comprehensive.

The potential advantage of **and**-MS over **or**-MS is that we can mark ghost k-mers with ones at some occurrences and therefore obtain masks with more ones in them, which could be beneficial, for instance, for additional improvements in mask compressibility.