Function-Assigned Masked Superstrings as a Versatile and Compact Data Type for k-Mer Sets

Ondřej Sladký[1], Pavel Veselý[1], and Karel Břinda[2]

[1] Computer Science Institute of Charles University, Prague, Czech Republic
ondra.sladky@gmail.com (OS), vesely@iuuk.mff.cuni.cz (PV)
[2] Inria, Irisa, Univ. Rennes, 35042 Rennes, France karel.brinda@inria.fr

Abstract. The exponential growth of DNA sequencing data calls for novel space-efficient algorithms for their compression and search. State-of-the-art approaches often use k-merization for data tokenization, yet efficiently representing and querying k-mer sets remains a significant bioinformatics challenge. Our recent work introduced the concept of masked superstrings, which compactly represent k-mer sets without reliance on common structural assumptions. However, the applicability of masked superstrings for set operations and membership queries remained open. Here, we develop the f-masked superstring framework, which integrates demasking functions f, enabling efficient k-mer set operations through concatenation. Combined with a tailored version of the FM-index, this framework provides a versatile, compact data structure for k-mer sets. We demonstrate its effectiveness with the FMSI program, which, when evaluated on bacterial pan-genomes, improves space efficiency by a factor of 1.4 to 4.5 compared to leading single k-mer-set indexing methods such as SSHash and SBWT. Overall, our work highlights the potential of f-masked superstrings as a versatile elementary data type for k-mer sets.

Keywords: Sequence bioinformatics · k-mer sets · Data structures · FM-index · Masked superstrings

1 Introduction

The exponential growth of DNA sequencing data calls for novel sublinear approaches for their compression and search [41, 63]. To address the increasing complexity of data, modern bioinformatics methods are increasingly relying on k-mer tokenization for data storage and analysis. This approach allows for a unified representation of various types of genomic data, such as sequencing reads, transcripts, assemblies, or species pan-genomes, as well as data reduction via techniques such as subsampling [64] and sketching [49]. Notable applications of k-mer-based methods include large-scale data search [5, 8, 14, 32], metagenomic classification [15, 64], infectious disease diagnostics [9, 13], and transcript abundance quantification [10, 51]. As the resulting sets can encompass hundreds of billions of k-mers [32], their efficient storage and associated set operations have become critical questions in sequence bioinformatics [43].

State-of-the-art data structures for k-mer sets increasingly use textual representations [42]. Originally, specialized methods for exact k-mer set representation drew ideas from information theory [20], which, however, provided unsatisfactory guarantees for the practically encountered k-mer sets. Subsequent approaches have focused on improving space requirements either through inexactness or by exploiting k-mer non-independence [16]. Particularly advantageous has been the observation that genomic k-mer sets are typically generated by a small number of long strings, a property referred to as the so-called *spectrum-like property (SLP)* [17]. If such strings are inferred from a given k-mer set, they can serve as its exact representation.

We refer to such textual representations as *(r)SPSS, (repetitive) Spectrum Preserving String Sets*. The first were *unitigs* [18, 19], which, originating from genome assembly, correspond to the non-branching paths of the associated vertex-centric de Bruijn graphs. Further compression has been achieved by so-called *simplitigs/SPSS* [11, 12, 54, 55], which permit arbitrary disjoint paths in the de Bruijn graphs, computed near-optimally using simple greedy algorithms [12, 55] or even optimally, still in linear time [59]. *Matchtigs/rSPSS* [58, 60] relax the requirement of paths to be disjoint, thereby further improving compression under SLP.

(r)SPSS are highly space-efficient, as long as the data are not derived from reduction techniques. (r)SPSS possess three properties defining their practicality: efficient computability, ease of compression, and support for efficient membership queries. When stored as text files, (r)SPSS can be easily compressed with general or specialized compressors [12, 55]. They can also be converted into efficient membership data structures by combining them with a full-text index [12, 18, 27, 55] or a minimal perfect hash function [4, 45, 52]. However, as $(k-1)$-long overlaps between k-mers are necessary for forming longer strings, (r)SPSS lose space efficiency for data obtained from SLP-breaking reduction techniques such as subsampling [64] or sketching [49].

We have recently tackled the challenge of general k-mer set representation via the concept of *masked superstrings (MS)* [62]. To remove the dependency on SLP and exploit k-mer overlaps of all lengths, we have shifted k-mer set modelization from de Bruijn graphs to more general overlap graphs. An MS consists of a superstring of the input k-mers and a mask indicating k-mer presence to avoid false positives. MS can be viewed a generalization of (r)SPSS, with improved compression capabilities and retained efficient computation [62]. However, efficient queries and set operations with MS-represented k-mer sets have remained an open problem, which we address in this paper.

Here, we develop the concept of *f-masked superstrings (f-MS)*, which combine MS with demasking functions f and concatenation operations, to create a versatile data type for k-mer sets. We show that the resulting algebraic structure is sufficiently general to support operations such as union, intersection, and symmetric difference. To also support k-mer queries, we develop an f-MS-tailored variant of the FM index [25], implement it in a program called FMSI and demonstrate the improvement over leading k-set-indexing methods.

1.1 Related Work

A large body of work focused on data structures for k-mer sets and their collections; we refer to [17, 43] for recent surveys. One approach for individual k-mer sets is to combine the aforementioned textual k-mer set representations based on the de Bruijn graph of the set [12, 18, 55, 60], i.e., (r)SPSS, with efficient string indexes such as the FM index [25] or BWA [36–39] (see also ProPhex [56, 57]). Another approach based on BWT is the BOSS data structure [7] that was implemented as an index for a collection of k-mer sets in VARI [48], VARI-merge [47], and Metagraph [32]. Taking BOSS as an inspiration, Spectral Burrows-Wheeler Transform (SBWT) [1] has been proposed as a compact representation of the de Bruijn graph, together with various approaches for its indexing. Themisto [2] further builds on SBWT and provides an index for collections of k-mer sets.

Hashing techniques have also been successful in the design of k-mer data structures. In particular, Bifrost [29] uses hash tables for indexing colored de Bruijn graphs, which encode collections of k-mer sets. Other popular data structures, such as BBHash [40], BLight [45], and SSHash [52], employ minimal perfect hash functions and serve as a base for constructing indexes such as FDBG [21], REINDEER [44], Fulgor [24], and pufferfish2 [23].

While the aforementioned approaches yield exact data structures, further space compression may be achieved by employing probabilistic techniques and allowing for a certain low-probability error, such as false positives. Namely, the counting quotient filter was used for k-mers in Squeakr [50], and this data structure was extended to an efficient index called dynamic Mantis [3]. Another line of work, e.g., [5, 8, 28, 35], employs variants of the Bloom filter to further reduce space requirements. Recently, Invertible Bloom Lookup Tables combined with subsampling have been used to estimate the Jaccard similarity coefficient [61].

Very recently, the Conway-Bromage-Lyndon (CBL) structure [46] builds on the work of Conway and Bromage [20] and combines smallest cyclic rotations of k-mers with sparse bit-vector encodings, to yield a dynamic and exact k-mer index supporting set operations, such as union, intersection, and difference. Finally, we note that set operations can also be carried out using some k-mer lists and counters, e.g., [31, 33]; however, these methods are unable to exploit structural properties of k-mer sets such as SLP.

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, 54, 55] if (1) it generates K and (2) each k-mer occurs only once among the simplitigs (uniqueness). Repetitive Spectrum Preserving String Sets (rSPSS) [58, 60], 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) [62] consists of a pair (S, M), where S is an arbitrary superstring of the k-mers in K and M is a binary mask M 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, and OFF otherwise. The set of k-mers generated by S are referred to as the *appearing k-mers*, and are further classified k-mers 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). MS are stored user 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 $K = \{\texttt{ACG}, \texttt{GGG}\}$. 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 $\{\texttt{ACG}, \texttt{GGG}\}$.

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 1. *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.,*

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

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, in Example 1, for the k-mer $Q = \texttt{GGG}$, it holds that $\lambda(S, M, Q) = (0, 1)$, as the first occurrence

Function f	**Definition**	**Compreh.**	**Use cases**
or	1 if $\lvert\lambda\rvert_1 > 0$ 0 if $\lvert\lambda\rvert_1 = 0$	yes	• MS (Sec. 3) and (r)SPSS (App. A) • f for union (Sec. 4)
xor	1 if $\lvert\lambda\rvert_1$ is odd 0 if $\lvert\lambda\rvert_1$ is even	yes	• f for sym. difference (Sec. 4)
and	1 if $\lambda \neq \epsilon \wedge \lvert\lambda\rvert_0 = 0$ 0 if $\lambda = \epsilon \vee \lvert\lambda\rvert_0 > 0$	yes	• allowing ON occurrences for ghost k-mers (App. C)
$[a,b]$-**threshold** ($1 \leq a \leq b$)	1 if $a \leq \lvert\lambda\rvert_1 \leq b$ 0 otherwise	iff $a = 1$	• f_o for intersection (Sec. 4) • f_o for set difference (Sec. 4)
one-or-nothing	1 if $\lvert\lambda\rvert_1 = 1$ 0 if $\lvert\lambda\rvert_1 = 0$ `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 $\lvert\lambda\rvert_1 = 2$ 0 if $\lvert\lambda\rvert_1 = 0$ `invalid` otherwise	no	• f_2 for set difference (Sec. 4)
all-or-nothing	1 if $\lambda \neq \epsilon \wedge \lvert\lambda\rvert_0 = 0$ 0 if $\lvert\lambda\rvert_1 = 0$ `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 3.

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

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

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 2. *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 Q should be encodable 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 3. *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 4. *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* $f(\lambda(S, M, Q)) = \texttt{invalid}$, *we call the* f-*MS* invalid.

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 [62], the most expensive part of MS computation (and thus also f-MS) is finding a short superstring. It is therefore natural to ask, which operations f-MS allows without its recomputation.

Lemma 1 (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 3.

Similarly, we can change the demasking functions f only by recoding the mask accordingly.

Lemma 2 (recoding). *Let* (f, S, M) *be a valid* f-*MS representing a* k-*mer set* K. *Then, for every comprehensive demasking functions* $f^{'}$, *there exists a valid mask* $M^{'}$ *such that* $(f^{'}, S, M^{'})$ *represents* K.

Proof. The existence of mask $M^{'}$ follows directly from Lemma 1 for the k-mer set K represented by $(f^{'}, S, M^{'})$ and the superstring S.

Example 2. Consider Example 1 with the set of 3-mers $K = \{\texttt{ACG}, \texttt{GGG}\}$, 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 1 and thus `GGG` is represented; however, **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, for instance, to `AcgGgg`. For functions that are not comprehensive, such recording may not impossible.

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. We deal with practical aspects of efficient implementation of this concept in Section 5.

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 5. *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 5 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 6. *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. We refer to them as *boundary k-mers* and to the occurrences of appearing k-mers of $S_1 + S_2$ that overlap both input superstrings as *boundary occurrences*.

4.2 Function-Preserving Set Operations

Union. As implicitly shown in [62], 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🐫 [62].

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 $S_N \subseteq \{0, 1, ..., N\}$ 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 $a \in S$. The sets S_N for $N = 1, 2, ...$ 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 3, 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 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 $(1, 1)$-MS,
- represent B using a $(2, 2)$-MS, and
- interpret the result as a $(1, 1)$-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)-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 $(2, 2)$ 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 FMS-index for Querying and Set Operations with f-MS

The above developed f-MS algebraic framework can be turned into a standalone data type for k-mer sets by a combination with a suitable support data structure. Here, we demonstrate this using a new index, called FMS-index, which is an adaptation of the FM-index [25] for f-MS, supporting masks and accelerating query operations.

FMS-index. The index is similar the standard FM-index [25] as typically used in bioinformatics, although it differs in two aspects. First, while it index the superstring as a standard string, all matching operations are performed directly in the suffix array coordinates; therefore, it omits the sampled suffix array as it is no longer required. Second, it adds an auxiliary table for the mask stored in the suffix array coordinates.

Specifically, FMS-index thus stores the BWT image of the superstring and a *SA-transformed mask*, whose i-th symbol corresponds to the mask symbol at the starting position of the i-th lexicographically smallest suffix of the superstring. Since the SA-transformed mask can be computed alongside the construction of the superstring FM-index, the mask transformation requires only linear time.

Set operations. In Appendix D, we describe how it is possible to implement the set operations as abstractly described using f-MS concatenation in Section 4, where we implement the concatenation via index merging. Additionally, to modify f without changing the represented k-mer set, we use function-specific recoding, which for comprehensive f is always guaranteed to exist from Lemma 2.

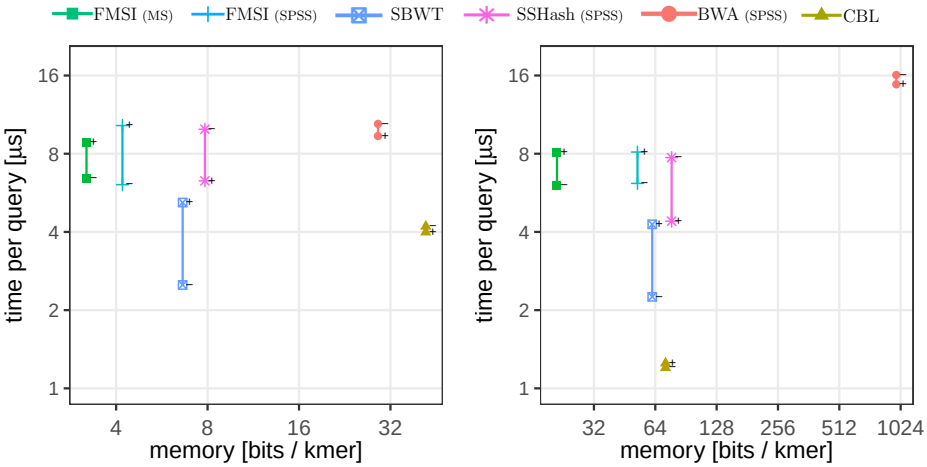
Membership queries. Querying a k-mer Q on an indexed f-MS consists of two steps. First, the suffix array interval of Q (and its reverse complement) for the superstring is identified using standard backward search [25]. Second, Q's presence or absence is decided based on the number of `0`s and `1`s (f is always symmetric by Definition 2) corresponding to this interval, which we obtain by rank queries on the transformed mask.

The first step can be performed in $O(k)$ time. The complexity of the second step depends on how efficiently we can evaluate f; for functions satisfying the property (P4) from Definition 3, the time complexity is constant.

6 Implementation and Experimental Evaluation

We implemented the FMS-index in a tool called FMSI, which can index MS, answer membership queries, and perform set operations. FMSI is available at GitHub (`https://github.com/OndrejSladky/fmsi`).

We compared FMSI against state-of-the-art single k-mer set indexes such as BWA [38], SBWT [1], SSHash [52], and CBL [46] and evaluated them on *E. coli* pan-genome (obtained as a k-mer union over *E. coli* genomes from the



(a) No subsampling ($r = 1.0$) **(b)** Uniformly subsampled ($r = 0.1$)

Fig. 1: Time and memory efficiency of k-mer membership queries. The plot compares time per query and memory per k-mer as measured for positive (+) and negative (-) queries for (a) full and (b) subsampled k-mer sets. The evaluation was performed using the complete 661k *E. coli* pan-genome [6], comprising all distinct canonical 23-mers (n=143M) across all *E. coli* genomes (n=89,950) in the collection. For BWA and SSHash, k-mers were passed as simplitigs computed by ProphAsm, for FMSI, k-mers were passed as masked superstrings computed by KmerCamel🐫 (global for MS and local with $d_{\max} = 1$ for SPSS; the latter is equivalent to ProphAsm [12],).

661k collection [6]). We provide more details on the methodology in Appendix E and provide supplementary material in the supplementary repository (`https://github.com/OndrejSladky/f-masked-superstrings-supplement`).

The results on the *E. coli* pan-genome for $k = 23$ without subsampling and with subsampling at rate 0.1 are presented in Figure 1.

Across all of the datasets, values of k, and subsampling rates, FMSI run on the MS computed by KmerCamel🐫's global greedy for pan-genomes with subsampling rate at least 0.1 required 1.4-4.5 times less memory for processing queries than all of the other methods, beating CBL and BWA up to 38 times, and attaining around 3-4 bits per k-mer on non-subsampled *E. coli* pan-genome, while still having competitive time performance. We note that the relative query time performance of SSHash in our experiments depended significantly on the size of k and that while the memory usage per k-mer grows with decreasing subsampling rate, the query time remains roughly the same for all algorithms.

7 Conclusion and Outlook

We have proposed f-masked superstrings (f-MS) as an abstract data type for k-mer sets that allows for seamless execution of set operations. It is primarily based on equipping masked superstrings (MS) from [62] with a demasking function f, and we have studied the effect of using several natural demasking functions. For symmetric set operations (e.g., union, intersection, symmetric difference), our approach is efficient even on a larger number of input sets as it reduces to simple concatenation of MS and possibly changing the demasking function. We have also shown how to perform asymmetric operations (e.g., set difference) in linear time on a constant number of input sets. However, from a practical point of view, the algorithm for asymmetric operations appears to be less efficient as it requires concatenating multiple copies of MS for some input sets. We leave open how to perform set difference and other asymmetric operations in a more efficient way.

Next, we have combined f-MS with FMS-Index, an adaptation of the FM index, and shown how to perform basic operations on the index and process k-mer membership queries efficiently. To demonstrate practicality, we have provided a prototype implementation in a tool called FMSI. While our experimental evaluation already shows significant reduction of memory usage, it gives no improvements in terms of time for query. However, we believe that FMSI can be further optimized. Furthermore, we leave as an open question how to design efficient algorithms for mask recasting and compaction without the need to export the f-MS.

Our work opens up several research directions for future investigation. First, our main focus was on processing individual k-mer sets that can be combined via set operations and queried for individual k-mers. The support for streaming querying of consecutive k-mers in a sequence can be added by using an adaptation of the bi-directional FM-index [34]. Another significant direction is to extend f-MS for indexing large collections of k-mer sets (reviewed in [43]). Finally, our focus was on set operations, while we leave maintaining k-mer sets under insertions and deletions to future work.

In conclusion, our long-term goal is a space- and time-efficient library for analyzing k-mer sets that includes all of these features, and we believe that the f-masked superstring framework is a useful step towards designing appropriate data structures for this library.

Acknowledgments. Ondřej Sladký and Pavel Veselý were partially supported by GA ČR project 22-22997S. Pavel Veselý was also partially supported by Center for Foundations of Modern Computer Science (Charles University project UNCE 24/SCI/008).

References

1. Alanko, J.N., Puglisi, S.J., Vuohtoniemi, J.: Small searchable κ-spectra via subset rank queries on the spectral Burrows-Wheeler transform. In: SIAM Conference on

Applied and Computational Discrete Algorithms (ACDA23), pp. 225–236, SIAM (2023), https://doi.org/10.1137/1.9781611977714.20

2. Alanko, J.N., Vuohtoniemi, J., Mäklin, T., Puglisi, S.J.: Themisto: a scalable colored k-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes. Bioinformatics **39**(Supplement_1), i260–i269 (2023), https://doi.org/10.1093/bioinformatics/btad233
3. Almodaresi, F., Khan, J., Madaminov, S., Ferdman, M., Johnson, R., Pandey, P., Patro, R.: An incrementally updatable and scalable system for large-scale sequence search using the Bentley-Saxe transformation. Bioinformatics **38**(12), 3155–3163 (2022), https://doi.org/10.1093/bioinformatics/btac142
4. Almodaresi, F., Sarkar, H., Srivastava, A., Patro, R.: A space and time-efficient index for the compacted colored de bruijn graph. Bioinformatics **34**(13), i169–i177 (Jun 2018), ISSN 1367-4811, https://doi.org/10.1093/bioinformatics/bty292, URL http://dx.doi.org/10.1093/bioinformatics/bty292
5. Bingmann, T., Bradley, P., Gauger, F., Iqbal, Z.: 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, pp. 285–303, Springer (2019), https://doi.org/10.1007/978-3-030-32686-9_21
6. Blackwell, G.A., Hunt, M., Malone, K.M., Lima, L., Horesh, G., Alako, B.T.F., Thomson, N.R., Iqbal, Z.: Exploring bacterial diversity via a curated and searchable snapshot of archived dna sequences. PLOS Biology **19**(11), 1–16 (11 2021), https://doi.org/10.1371/journal.pbio.3001421
7. Bowe, A., Onodera, T., Sadakane, K., Shibuya, T.: Succinct de Bruijn graphs. In: Raphael, B.J., Tang, J. (eds.) Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7534, pp. 225–235, Springer (2012), https://doi.org/10.1007/978-3-642-33122-0_18
8. Bradley, P., Den Bakker, H.C., Rocha, E.P., McVean, G., Iqbal, Z.: Ultrafast search of all deposited bacterial and viral genomic data. Nature Biotechnology **37**(2), 152–159 (2019)
9. Bradley, P., Gordon, N.C., Walker, T.M., Dunn, L., Heys, S., Huang, B., Earle, S., Pankhurst, L.J., Anson, L., De Cesare, M., et al.: Rapid antibiotic-resistance predictions from genome sequence data for staphylococcus aureus and mycobacterium tuberculosis. Nature Communications **6**(1), 10063 (2015), https://doi.org/10.1038/ncomms10063
10. Bray, N.L., Pimentel, H., Melsted, P., Pachter, L.: Near-optimal probabilistic RNA-seq quantification. Nature Biotechnology **34**(5), 525–527 (2016), https://doi.org/10.1038/nbt.3519
11. Břinda, K.: Novel computational techniques for mapping and classification of Next-Generation Sequencing data. PhD thesis, Université Paris-Est (2016), https://doi.org/10.5281/zenodo.1045317
12. Břinda, K., Baym, M., Kucherov, G.: Simplitigs as an efficient and scalable representation of de Bruijn graphs. Genome Biology **22**(96) (2021), https://doi.org/10.1186/s13059-021-02297-z
13. Břinda, K., Callendrello, A., Ma, K.C., MacFadden, D.R., Charalampous, T., Lee, R.S., Cowley, L., Wadsworth, C.B., Grad, Y.H., Kucherov, G., et al.: Rapid inference of antibiotic resistance and susceptibility by genomic neighbour typing. Nature Microbiology **5**(3), 455–464 (2020), https://doi.org/10.1038/s41564-019-0656-6
14. Břinda, K., Lima, L., Pignotti, S., Quinones-Olvera, N., Salikhov, K., Chikhi, R., Kucherov, G., Iqbal, Z., Baym, M.: Efficient and robust search of microbial

genomes via phylogenetic compression. bioRxiv **2023.04.15.536996** (2023), https://doi.org/10.1101/2023.04.15.536996
15. Břinda, K., Salikhov, K., Pignotti, S., Kucherov, G.: Prophyle 0.3.1.0. Zenodo **5281** (2017), https://doi.org/10.5281/zenodo.5237391, URL https://prophyle.github.io
16. Chikhi, R.: K-mer data structures in sequence bioinformatics. HDR thesis, Institut Pasteur Ecole Doctorale "EDITE" (2021), URL http://rayan.chikhi.name/pdf/RChikhi-HDR.pdf
17. Chikhi, R., Holub, J., Medvedev, P.: Data structures to represent a set of k-long DNA sequences. ACM Computing Surveys **54**(1), 17:1–17:22 (2022), https://doi.org/10.1145/3445967
18. Chikhi, R., Limasset, A., Jackman, S., Simpson, J.T., Medvedev, P.: On the representation of de Bruijn graphs. In: Sharan, R. (ed.) Research in Computational Molecular Biology, pp. 35–55, Springer International Publishing, Cham (2014), ISBN 978-3-319-05269-4, https://doi.org/10.1007/978-3-319-05269-4_4
19. Chikhi, R., Limasset, A., Medvedev, P.: Compacting de Bruijn graphs from sequencing data quickly and in low memory. Bioinformatics **32**(12), i201–i208 (2016), ISSN 1367-4803, https://doi.org/10.1093/bioinformatics/btw279
20. Conway, T.C., Bromage, A.J.: Succinct data structures for assembling large genomes. Bioinformatics **27**(4), 479–486 (2011), https://doi.org/10.1093/bioinformatics/btq697
21. Crawford, V.G., Kuhnle, A., Boucher, C., Chikhi, R., Gagie, T.: Practical dynamic de Bruijn graphs. Bioinformatics **34**(24), 4189–4195 (2018), https://doi.org/10.1093/bioinformatics/bty500
22. Croucher, N.J., Finkelstein, J.A., Pelton, S.I., Parkhill, J., Bentley, S.D., Lipsitch, M., Hanage, W.P.: Population genomic datasets describing the post-vaccine evolutionary epidemiology of streptococcus pneumoniae. Scientific Data **2** (2015), https://doi.org/10.1038/sdata.2015.58
23. Fan, J., Khan, J., Pibiri, G.E., Patro, R.: Spectrum preserving tilings enable sparse and modular reference indexing. In: Tang, H. (ed.) Research in Computational Molecular Biology - 27th Annual International Conference, RECOMB 2023, Istanbul, Turkey, April 16-19, 2023, Proceedings, Lecture Notes in Computer Science, vol. 13976, pp. 21–40, Springer (2023), https://doi.org/10.1007/978-3-031-29119-7_2
24. Fan, J., Khan, J., Singh, N.P., Pibiri, G.E., Patro, R.: Fulgor: a fast and compact k-mer index for large-scale matching and color queries. Algorithms for Molecular Biology **19**(1), 3 (2024), https://doi.org/10.1186/S13015-024-00251-9
25. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: Proceedings 41st Annual Symposium on Foundations of Computer Science, SFCS-00, IEEE Comput. Soc (2000), https://doi.org/10.1109/sfcs.2000.892127
26. Gog, S., Beller, T., Moffat, A., Petri, M.: From theory to practice: Plug and play with succinct data structures. In: 13th International Symposium on Experimental Algorithms, (SEA 2014), pp. 326–337 (2014)
27. Guo, H., Fu, Y., Gao, Y., Li, J., Wang, Y., Liu, B.: degsm: Memory scalable construction of large scale de bruijn graph. IEEE/ACM Transactions on Computational Biology and Bioinformatics **18**(6), 2157–2166 (Nov 2021), ISSN 2374-0043, https://doi.org/10.1109/tcbb.2019.2913932, URL http://dx.doi.org/10.1109/TCBB.2019.2913932
28. Gupta, G., Yan, M., Coleman, B., Kille, B., Elworth, R.A.L., Medini, T., Treangen, T., Shrivastava, A.: 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, p. 2226–2234, SIGMOD '21, Association for Computing Machinery, New York, NY, USA (2021), ISBN 9781450383431, https://doi.org/10.1145/3448016.3457333, URL https://doi.org/10.1145/3448016.3457333

29. Holley, G., Melsted, P.: Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. Genome Biology **21**(1), 1–20 (2020), https://doi.org/10.1186/s13059-020-02135-8
30. Holt, J., McMillan, L.: Merging of multi-string bwts with applications. Bioinformatics **30**(24), 3524–3531 (Aug 2014), ISSN 1367-4803, https://doi.org/10.1093/bioinformatics/btu584
31. Kaplinski, L., Lepamets, M., Remm, M.: 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), ISSN 2047-217X, https://doi.org/10.1186/s13742-015-0097-y
32. Karasikov, M., Mustafa, H., Danciu, D., Zimmermann, M., Barber, C., Rätsch, G., Kahles, A.: Indexing all life's known biological sequences. bioRxiv **2020.10.01.322164** (2024), https://doi.org/10.1101/2020.10.01.322164
33. Kokot, M., Długosz, M., Deorowicz, S.: KMC 3: counting and manipulating k-mer statistics. Bioinformatics **33**(17), 2759–2761 (05 2017), ISSN 1367-4803, https://doi.org/10.1093/bioinformatics/btx304
34. Lam, T.W., Li, R., Tam, A., Wong, S., Wu, E., Yiu, S.M.: High throughput short read alignment via bi-directional bwt. In: 2009 IEEE International Conference on Bioinformatics and Biomedicine, IEEE (2009), https://doi.org/10.1109/bibm.2009.42
35. Lemane, T., Medvedev, P., Chikhi, R., Peterlongo, P.: kmtricks: efficient and flexible construction of Bloom filters for large sequencing data collections. Bioinformatics Advances **2**(1), vbac029 (04 2022), ISSN 2635-0041, https://doi.org/10.1093/bioadv/vbac029
36. Li, H.: Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. Bioinformatics **28**(14), 1838–1844 (2012), ISSN 1367-4803, https://doi.org/10.1093/bioinformatics/bts280
37. Li, H.: Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv preprint arXiv:1303.3997 (2013), https://doi.org/10.48550/arXiv.1303.3997
38. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics **25**(14), 1754–1760 (2009), https://doi.org/10.1093/bioinformatics/btp324
39. Li, H., Durbin, R.: Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics **26**(5), 589–595 (2010), https://doi.org/10.1093/bioinformatics/btp698
40. Limasset, A., Rizk, G., Chikhi, R., Peterlongo, P.: Fast and scalable minimal perfect hashing for massive key sets. In: Iliopoulos, C.S., Pissis, S.P., Puglisi, S.J., Raman, R. (eds.) 16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK, LIPIcs, vol. 75, pp. 25:1–25:16, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017), https://doi.org/10.4230/lipics.sea.2017.25
41. Loh, P.R., Baym, M., Berger, B.: Compressive genomics. Nature Biotechnology **30**(7), 627–630 (Jul 2012), ISSN 1546-1696, https://doi.org/10.1038/nbt.2241, URL http://dx.doi.org/10.1038/nbt.2241
42. Marchet, C.: Sneak peek at the tig sequences: useful sequences built from nucleic acid data (2022), https://doi.org/10.48550/arXiv.2209.06318

43. Marchet, C., Boucher, C., Puglisi, S.J., Medvedev, P., Salson, M., Chikhi, R.: Data structures based on k-mers for querying large collections of sequencing data sets. Genome Research **31**(1), 1–12 (2021), https://doi.org/10.1101/gr.260604.119
44. Marchet, C., Iqbal, Z., Gautheret, D., Salson, M., Chikhi, R.: REINDEER: efficient indexing of k-mer presence and abundance in sequencing datasets. Bioinformatics **36**(Supplement-1), i177–i185 (2020), https://doi.org/10.1093/bioinformatics/btaa487
45. Marchet, C., Kerbiriou, M., Limasset, A.: Blight: efficient exact associative structure for k-mers. Bioinformatics **37**(18), 2858–2865 (2021), https://doi.org/10.1093/bioinformatics/btab217
46. Martayan, I., Cazaux, B., Limasset, A., Marchet, C.: Conway-Bromage-Lyndon (CBL): an exact, dynamic representation of k-mer sets. bioRxiv **2024.01.29.577700** (2024), https://doi.org/10.1101/2024.01.29.577700
47. Muggli, M.D., Alipanahi, B., Boucher, C.: Building large updatable colored de Bruijn graphs via merging. Bioinformatics **35**(14), i51–i60 (2019), https://doi.org/10.1093/bioinformatics/btz350
48. Muggli, M.D., Bowe, A., Noyes, N.R., Morley, P.S., Belk, K.E., Raymond, R., Gagie, T., Puglisi, S.J., Boucher, C.: Succinct colored de Bruijn graphs. Bioinformatics **33**(20), 3181–3187 (2017), https://doi.org/10.1093/bioinformatics/btx067
49. Ondov, B.D., Treangen, T.J., Melsted, P., Mallonee, A.B., Bergman, N.H., Koren, S., Phillippy, A.M.: Mash: fast genome and metagenome distance estimation using minhash. Genome Biology **17**(1) (Jun 2016), ISSN 1474-760X, https://doi.org/10.1186/s13059-016-0997-x, URL http://dx.doi.org/10.1186/s13059-016-0997-x
50. Pandey, P., Bender, M.A., Johnson, R., Patro, R.: Squeakr: an exact and approximate k-mer counting system. Bioinformormatics **34**(4), 568–575 (2018), https://doi.org/10.1093/bioinformatics/btx636
51. Patro, R., Duggal, G., Love, M.I., Irizarry, R.A., Kingsford, C.: Salmon provides fast and bias-aware quantification of transcript expression. Nature Methods **14**(4), 417–419 (2017), https://doi.org/10.1038/nmeth.4197
52. Pibiri, G.E.: Sparse and skew hashing of K-mers. Bioinformatics **38**(Supplement_1), i185–i194 (2022), ISSN 1367-4803, https://doi.org/10.1093/bioinformatics/btac245
53. Pibiri, G.E.: On weighted k-mer dictionaries. Algorithms for Molecular Biology **18**(1) (Jun 2023), ISSN 1748-7188, https://doi.org/10.1186/s13015-023-00226-2, URL http://dx.doi.org/10.1186/s13015-023-00226-2
54. Rahman, A.: Compression algorithms for de Bruijn graphs and uncovering hidden assembly artifacts. PhD thesis, The Pennsylvania State University (2023)
55. Rahman, A., Medevedev, P.: Representation of k-mer sets using spectrum-preserving string sets. Journal of Computational Biology **28**(4), 381–394 (2021), https://doi.org/10.1089/cmb.2020.0431, pMID: 33290137
56. Salikhov, K.: Efficient algorithms and data structures for indexing dna sequence data. PhD thesis, Université Paris-Est (2017)
57. Salikhov, K., Břinda, K., Pignotti, S., Kucherov, G.: ProPhex. https://github.com/prophyle/prophex (2018), https://doi.org/10.5281/zenodo.1247432
58. Schmidt, S.: Unitigs are not enough: the advantages of superunitig-based algorithms in bioinformatics. PhD thesis, University of Helsinki (2023)
59. Schmidt, S., Alanko, J.N.: Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time. Algorithms for Molecular Biology **18**(1), 5 (2023), https://doi.org/10.1186/s13015-023-00227-1

60. Schmidt, S., Khan, S., Alanko, J.N., Pibiri, G.E., Tomescu, A.I.: Matchtigs: minimum plain text representation of k-mer sets. Genome Biology **24**(1), 136 (2023), ISSN 1474-760X, https://doi.org/10.1186/s13059-023-02968-z
61. Shibuya, Y., Belazzougui, D., Kucherov, G.: Efficient Reconciliation of Genomic Datasets of High Similarity. In: Boucher, C., Rahmann, S. (eds.) 22nd International Workshop on Algorithms in Bioinformatics (WABI 2022), Leibniz International Proceedings in Informatics (LIPIcs), vol. 242, pp. 14:1–14:14, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022), ISBN 978-3-95977-243-3, ISSN 1868-8969, https://doi.org/10.4230/LIPIcs.WABI.2022.14
62. Sladký, O., Veselý, P., Břinda, K.: Masked superstrings as a unified framework for textual k-mer set representations. bioRxiv **2023.02.01.526717** (2023), https://doi.org/10.1101/2023.02.01.526717
63. Stephens, Z.D., Lee, S.Y., Faghri, F., Campbell, R.H., Zhai, C., Efron, M.J., Iyer, R., Schatz, M.C., Sinha, S., Robinson, G.E.: Big data: Astronomical or genomical? PLoS Biology **13**(7), e1002195 (2015), https://doi.org/10.1371/journal.pbio.1002195
64. Wood, D.E., Salzberg, S.L.: Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biology **15**(3), 1–12 (2014), https://doi.org/10.1186/gb-2014-15-3-r46

A Proof of Uniqueness of Union Function

Masked superstrings [62], 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 [62] 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 1. ***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 3, 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 2. ***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 1.

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

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 3. *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 4. *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.

This representation has its clear benefits. Most importantly, one can determine the presence or absence of a k-mer by looking at the mask at any occurrence of the k-mer. For example, this makes indexing of **all-or-nothing**-MS easier than indexing general f-MS, as we do not need to query the rank to determine the number of ON occurrences. Instead, we can simply determine the presence or absence of a k-mer based on any of its occurrences, as discussed in Section 5.

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.

We further note that **all-or-nothing**-MS can be viewed as **or**-MS that maximize the number of ones in the mask.

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.

D Set Operations on Indexed k-Mer Sets

In this appendix we provide several basic operations with the FMS-index introduced in Section 5 and describe how to perform set operations on f-MS with these building blocks. To implement operations for f-MS using this index, we need to decompose individual f-MS operations into primitive operations with the index. On a low level, these involve exporting the f-MS to its string representation from the index, f-MS concatenation, changing the demasking function, and compaction to decrease its size by data deduplication.

Exporting an f-MS to its string representation. To retrieve the f-MS in the string form from the index, it is sufficient to revert the Burrows-Wheeler transform using the LF mapping and translate the mask bits from suffix array coordinates to the string coordinates.

f-MS concatenation by index merging. We can export the f-MS, concatenate them in their string form and index the result, which achieves linear time

complexity. Alternatively, it is possible merge the indexes without exporting by directly merging the BWTs [30] and masks alongside it.

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 achieve in linear time using a single/two-pass algorithms, respectively [62]. 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 linear time, using two different approaches: One option is 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 [62]. We leave it to future work to design algorithms for compaction and for mask recasting directly in the FMS-index without the need to export the f-MS.

Set operations. Using an indexed f-MS, set operations such as union, (a)symmetric difference, or intersection can be performed directly via their associated abstract operations in Section 4. Indeed, we implement concatenation of MS via index merging. Prior to concatenating, we only need to ensure that each input set is represented using a correct demasking function as required by the operation (Table 1), and to recast the mask if it is not the case.

After the concatenation, depending on our use-case, we recast the mask if we need a different demasking function than the one resulting from individual operations. Finally, it may be desirable to compact the f-MS in case the resulting f-masked superstring is unnecessarily large for the set it represents; more precisely, when many k-mers appear in the f-MS multiple times. Given that we either perform a symmetric set operation or the number of input sets is constant, all of these steps can be implemented in linear time and thus, the total time complexity of each set operation is also linear.

E Benchmarking Methodology

In this appendix we give a high-level overview of our tool FMSI and provide details about the methods used for experiments with indexing (Section 6) as well as for experiments with set operations (Appendix F).

Implementation of f-MS operations in FMSI. We implemented indexing f-MS and the associated k-mer set operations in a tool called FMSI (f-Masked Superstring Index). The tool supports membership queries on indexed f-MS and further provides an implementation of basic building-block operations such as exporting, merging, and compaction that are used to perform set operations.

Index merging is implemented via export, concatenation of the underlying f-MS, and then reindexing. Compaction is implemented using k-mer counting and KmerCamel🐫 [62] to construct a superstring. All the demasking functions mentioned in Table 1 are supported directly by FMSI, and users can possibly add their custom ones.

FMSI was developed in C++ and is available from GitHub (`https://github.com/OndrejSladky/fmsi`) under the MIT license. The implementation is based on the sdsl-lite library [26], available at `https://github.com/simongog/sdsl-lite`, and also uses KmerCamel🐫, available at `https://github.com/OndrejSladky/kmercamel`.

Benchmarking methodology. We evaluated the performance of FMSI (version at commit `f5c8909`) on bacterial and viral pan-genomes and on a nematode genome, both in its efficiency of construction and querying as well as in performance on set operations, namely computing intersections, symmetric differences, and unions of k-mer sets. We measured the storage requirements for each index, both the time and memory requirements for construction and the time and memory requirements for queries with isolated k-mers. We tested both positive and negative queries; to generate positive queries, we took a random subset of 10^6 distinct k-mers from the queried dataset, and to obtain negative queries, we took a random subset of 10^6 distinct k-mers from a part of chromosome 1 of the human genome (genome assembly GRCh38.p14), excluding those in the queried dataset. All the algorithms were run on a single thread on a server with AMD EPYC 7302 (3 GHz) processor and 251 GB RAM.

Experimental evaluation of indexing. We compared time and memory requirements for processing both positive and negative queries of FMSI to state-of-the-art programs for indexing individual k-mer sets, namely:

- **BWA** [38] (`https://github.com/lh3/bwa`, commit `139f68f`), a state-of-the-art aligner based on the FM index; for processing queries, we used `bwa fastmap -w 999999` command [36], with the index computed from the simplitigs obtained using ProphAsm [12].
- **SBWT** [1] (`https://github.com/algbio/SBWT`, commit `c433b53`), an index based on the spectral Burrows-Wheeler transform; we used the default plain-matrix variant as it achieves the best query times in [1], added all reverse complements to the index, and disabled streaming queries support (which decreases memory usage). That is, we ran `sbwt build -m 10 --add-reverse-complements --no-streaming-support`. We note that SBWT required substantial disk space during index construction (up to tens of GBs for the *E. coli* pan-genome).
- **SSHash** [52, 53] (`https://github.com/jermp/sshash`, commit `11dc8f2`), an index based on minimal perfect hashing of k-mers, computed using `sshash build -m M -s S -l 11`, where we set the minimizer length to `M = k//2 + 1` and used seeds `S` equal to 1, 42, and 88. We note that SSHash may fail during the index computation, and we had to adjust the minimizer length and try multiple seeds `S` before we managed to compute indexes for all datasets, values of k, and subsampling rates.

– **CBL** [46] (`https://github.com/imartayan/CBL`, commit `b3a1158`), a very recent method based on smallest cyclic rotations of k-mers. The index was computed on canonical k-mers, to handle reverse complements, that is, we ran `cbl build -c`. We note that CBL needs to be compiled for every value of k separately.

We have run FMSI on the MS computed by KmerCamel🐫 [62], specifically the global and local greedy algorithms (local is run with $d = 1$).

This experiment was done using an *E. coli* pan-genome (obtained as a union of k-mers of *E. coli* genomes from the 661k collection [6]) and *S. pneumoniae* pan-genome (computed from 616 assemblies from a study of children in Massachusetts, USA [22]) and further verified on a *SARS-CoV-2* pan-genome (n=14.7 M, total genome length 430 Gbp). To verify the behavior across diverse datasets, we also provide experimental results for *subsampled* k-mer sets of these three pan-genomes; we note that after subsampling the spectrum-like property (SLP) no longer hold. Specifically, for a given subsampling rate $r \in [0, 1]$, we selected a uniformly random subset of $r \cdot N$ distinct k-mers of the original pan-genome, where N is the total number of k-mers of the pan-genome.

F Experimental Evaluation of Set Operations

In this appendix, we demonstrate the feasibility of using FMSI to perform set operations on k-mer sets with our prototype implementation of FMS-index, FMSI (see also Appendix E for details on FMSI). Our proposed pipeline for set operations, as depicted in Figure 2, 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. 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 (this can be done even before mask recasting). The last two steps are concatenating the two FMS-indexes by index merging and compacting the resulting FMS-index if needed. Note that once indexed, we can ask membership queries on the resulting k-mer sets.

For this experiment, we used genomes of *C. elegans* (`NC_003279.8`, 100M base pairs) and *C. briggsae* (`NC_013489.2`, 108M base pairs). 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 2. Overall, at every step, the memory required to query the indexed f-MS was around 3 bits per superstring character, which in case of the roundworms was almost the same as the number of k-mers. This trend continues even for the merged FMS-index, albeit for the compacted concatenated result the per-k-mer memory was higher as it was as low as latent memory required to run FMSI. Note also that the fact that compaction significantly

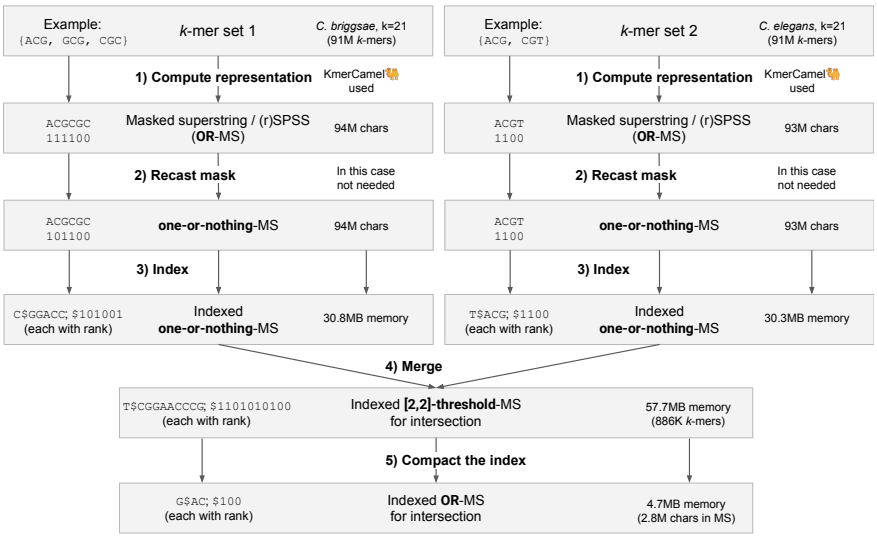


Fig. 2: Set operations workflow using f-masked superstrings: example on intersection. The workflow also contains illustrative example on a set of 3-mers as well as time for the operations and experimental data for running the workflow on *C. briggsae* and *C. elegans* genomes with $k = 21$. The experimental data contains the number of MS characters after each change, number of represented k-mers and for each indexed representation the memory required for quering the underlying set.

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 almost negligible length reduction. For data about symmetric difference and union as well as for other values of k, see the supplementary repository (`https://github.com/OndrejSladky/f-masked-superstrings-supplement`).