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

Ondřej Sladký
Computer Science Institute of Charles University, Prague, Czech Republic

Pavel Veselý
Computer Science Institute of Charles University, Prague, Czech Republic

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

Abstract

The exponential growth of genome databases calls for novel space-efficient algorithms for data compression and search. State-of-the-art approaches often rely on k-merization for data tokenization, yet efficiently representing and querying k-mer sets remains a significant challenge in bioinformatics. Our recent work has introduced the concept of masked superstring for compactly representing k-mer sets, designed without reliance on common structural assumptions on k-mer data. However, despite their compactness, the practicality of masked superstrings for set operations and membership queries was previously unclear. Here, we propose the f-masked superstring framework, which additionally integrates demasking functions f, enabling efficient k-mer set operations through concatenation. When combined with the FMS-index, a new index for f-masked superstrings based on a simplified FM-index, we obtain a versatile, compact data structure for k-mer sets. We demonstrate its power through the FMSI program, which, when evaluated on bacterial pan-genomic data, achieves memory savings of a factor of 3 to 10 compared to state-of-the-art single k-mer-set indexing methods such as SBWT and CBL. Our work presents a theoretical framework with promising practical advantages such as space-efficiency, demonstrating the potential of f-masked superstrings in k-mer-based methods as a generic data type.

2012 ACM Subject Classification Applied computing → Computational genomics; Applied computing → Bioinformatics

Keywords and phrases computational genomics, k-mer sets, masked superstrings, set operations, indexing

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplementary Material Data and pipelines used for experiments, as well as additional results and plots, are available at `https://github.com/OndrejSladky/f-masked-superstrings-supplement`.
Software (Source Code): `https://github.com/OndrejSladky/fmsi`

Funding *Ondřej Sladký*: Supported by GA ČR project 22-22997S.
Pavel Veselý: Supported by GA ČR project 22-22997S and by Center for Foundations of Modern Computer Science (Charles Univ. project UNCE 24/SCI/008).

1 Introduction

Storage and analyses of the exponentially growing collections of genomic data [41, 62] require novel sublinear approaches that could cope with highly heterogeneous data. Many modern bioinformatics methods use k-merization of input data as the central tokenization technique, since this allows to represented data of various forms such as assemblies, sequencing reads, genes, or transcripts in a unified form. Furthermore, k-mer sets can easily be combined with subsampling and sketching. Notable applications of k-mer-based methods include large-scale data search [7, 4, 30, 13], metagenomic classification [63, 14], infectious disease diagnostics [8, 12], and transcript abundance quantification [9, 50], to name at least a few. Moreover, k-mers find their use in the direct study of biological phenomena, such as in Genome-Wide Association Studies [34] or the studies of bacterial defense systems [59].

The widespread use of k-mer sets across computational biology calls for efficient techniques for their space-efficient storage and memory-efficient operations such as membership queries or k-mer set operations. However, the development of such methods is complicated by the enormous heterogeneity of k-mer sets encountered in practice. The size of k-mer sets can range from several tens to billions, with k-mer sizes from values below ten to more than 100 (the most typical value being 31).

As general information-theory-based methods provide unsatisfactory guarantees for the efficiency, methods based on inexactness or non-independence of k-mers have emerged [15]. In particular, typical k-mer sets encountered in genomic applications have the property that the k-mers are substrings of a small number of larger strings, typically referred to as the spectrum-like property (SLP) [16].

Textual representations of k-mer sets have been particularly efficient under SLP. *Simplitigs/Spectrum Preserving String Sets* (SPSS) [11, 10, 53, 52, 57] were introduced as a representation of k-mer sets via a set of strings such that each k-mer appears exactly once as a substring. Simplitigs/SPSS provide a more concise representations compared to classical *unitigs* [17, 18], which correspond to non-branching paths in the de Bruijn graph of the k-mer set. *Matchtigs* (a.k.a. *repetitive Spectrum Preserving String Sets*, or rSPSS) [58, 56] generalized over simplitigs/SPSS by only requiring that each k-mer appears in *at least* one of the strings, allowing for representations with even smaller number of strings.

All of these representations can be easily turned into efficient membership data structures by combining with a full-text index or a minimum perfect hash function. Likewise, they provide an efficient storage format when combined with a generalized or specialized data compressor. However, all of these representations still rely on $(k-1)$-long overlaps between k-mers that might not be present among k-mers, particularly in modern applications combining subsampling techniques with long-read sequencing data.

In our previous work [61], we have developed the concept of masked superstring for k-mer set representations. Masked superstrings are based on the overlap graph of k-mers, instead of the de Bruijn graph, and thus removing the dependency on the SLP. They can be seen as a generalization of representations above, thus unifying the theory and also the practice of their use. However, despite their generality and obvious compression capabilities, their application in k-mer based data structures remained an open problem, especially in the context of membership queries and k-mer set operations.

Here, we present a generalization of masked superstrings by adding additional flexibility of k-mer mask interpretation by different functions f, resulting in so-called f-masked superstrings. We characterize properties of functions f that can be used and demonstrate that f-masked superstrings provide a sufficiently general structure to be used as a standalone data type

for k-mer sets, supporting operations such as union, intersection, symmetric difference, and querying k-mers. Finally, we support our theoretical results with practical implementations of the individual algorithms and demonstrate that f-masked superstrings are more space-efficient compared to other existing representations.

1.1 Related Work

A large body of work focused on data structures for k-mer sets and their collections; we refer to [16, 42] 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 [17, 11, 53, 58], i.e., (r)SPSS, with efficient string indexes such as the FM index [24] or BWA [38, 39, 36, 37] (see also ProPhex [55, 54]). Another approach based on BWT is the BOSS data structure [6] that was implemented as an index for a collection of k-mer sets in VARI [47], VARI-merge [46], and Metagraph [30]. 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 [27] 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 [44], and SSHash [51], employ minimal perfect hash functions and serve as a base for constructing indexes such as FDBG [20], REINDEER [43], Fulgor [23], and pufferfish2 [22].

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 [49], and this data structure was extended to an efficient index called dynamic Mantis [3]. Another line of work, e.g., [7, 4, 26, 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 [60].

Very recently, the Conway-Bromage-Lyndon (CBL) structure [45] builds on the work of Conway and Bromage [19] 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., [29, 32]; however, these methods are unable to exploit structural properties of k-mer sets such as the SLP.

1.2 Our Contribution and Organization of the Paper

In this paper, we propose *function-assigned masked superstrings*, a data structure for k-mers, which fully leverages overlaps among the k-mers, while also supporting efficient set operations and indexing.

First, in **Section 3**, we develop the theoretical foundations of *function-assigned masked superstrings* (f-MS), which can be seen as an algebraic generalization of masked superstrings from our earlier work [61]. While our previous work [61] considered a k-long substring of K represented by (S, M) if any of its occurrences in S is unmasked, that is, the **or** of corresponding mask bits equals 1, here we use other functions f for determining the presence or absence of a k-mer from (S, M), allowing besides **or**-masked superstrings also **xor**-MS, **and**-MS, and many others types.

Then, in **Section 4**, we demonstrate that this formalism enables set operations such as union, intersection, and symmetric difference. In a nutshell, this reduces to just concatenating the corresponding f-masked superstrings and possibly changing the function f used (**Section 4.1**). While for union and symmetric difference, this is possible via a simple concatenation using **or** and **xor**, respectively, for both the input and the output f-MS (**Section 4.2**), intersection requires more careful treatment (**Section 4.3**). Indeed, we observe that *no function* f with certain natural properties corresponds to the intersection if the input and output masked superstrings use the same f (**Appendix B**). We circumvent this impossibility by transforming f alongside these operations (**Section 4.3**). We further show that the resulting scheme for intersection generalizes to *any* set operation on any number of sets. One key advantage of our approach is that a single masked superstring, obtained by concatenating masked superstrings of several datasets, can be used for indexing the results of *multiple* set operations, just by changing the function f.

Building on the algebraic framework of f-masked superstrings, in **Section 5** we design a standalone data type for k-mer sets. This data type is based on FMS-index, an adaptation of the FM index [24] to our framework (**Section 5.1**). We show how to efficiently process membership queries (**Section 5.3**) and perform basic operations, such as concatenation of f-MSes or compaction to remove redundant k-mers (**Section 5.2**), that allow for executing set operations (**Section 5.4**). Finally, in **Section 6**, we support our theoretical results with a prototype implementation that we use to demonstrate that f-masked superstrings are more space-efficient compared to other existing methods.

2 Preliminaries

Alphabet and strings. We consider strings over a constant-size alphabet Σ, typically the ACGT or binary alphabets. Let Σ^* be the set of all finite strings over Σ. For an empty sequence of alphabet letters, we use ϵ. By $|S|$ we denote the length of a string S and by $|S|_c$ we denote the number of occurrences of the letter c in S. For two strings S and T, let $S + T$ be their concatenation.

k-mers and their sets. We refer to strings of size k over the ACGT alphabet as k-mers and we typically denote by Q an individual k-mer and by K a set of k-mers. Further, we distinguish the uni-directional model and the bi-directional model where we consider k-mer and its reverse complement as equivalent. Unless explicitly stated otherwise, our results apply both in the uni-directional and bi-directional model, but for clarity we provide examples in the uni-directional model.

SPSS-based representations of k-mer sets. *Simplitigs/Spectrum Preserving String Sets* [11, 10, 53, 52] (SPSS) and *matchtigs* [58, 56] are formed by a set of strings such that every k-mer of these strings is in the represented set and vice versa. Thus, they correspond to a path cover of the de Bruijn graph of the set, with SPSS having the restriction that each k-mer appears exactly once. Since a k-mer may appear multiple times in matchtigs, we also refer to them as *Repetitive Spectrum Preserving String Sets* (rSPSS). We use (r)SPSS to denote any of these representations based on the de Bruijn graph.

Masked superstrings. A masked superstring is a pair (S, M) where S is a superstring and M an associated binary mask of the same length. We call all the k-mers that appear as a substring in S as the *appearing k-mers*. The appearing k-mers that are in the set represented by the masked superstring are called the *represented k-mers* and the remaining appearing k-mers are called *ghost k-mers*. For a given superstring S and a k-mer set K, we call a *compatible mask* every mask M such that (S, M) represents K. Furthermore, as a k-mer can

have multiple occurrences in the superstring, an occurrence is called ON if there is 1 at the corresponding position in the mask, and OFF otherwise. Note that being a represented k-mer then corresponds to having at least one ON occurrence.

Encoding conventions. The last $k-1$ positions of every mask are always set to 0. When discussing specific masked superstrings, we present them in their masked-cased superstring encoding (enc2 in [61]), where each character of the superstring is upper-case if it corresponds to 1 in the mask, and lower-case if it corresponds to 0.

▶ **Example 1.** Consider the set of 3-mers $K = \{\texttt{ACG}, \texttt{GGG}\}$ of the k-mers to be represented and the superstring ACGGGG resulting from their concatenation. There are three compatible masks – 101100, 101000 and 100100 – since each of the two k-mers must have at least one ON occurrence, and the occurrence of the ghost k-mer CGG must always be OFF. With the last mask, the masked-cased encoding would be AcgGgg. Conversely, when parsing AcgGgg as a masked superstring for $k = 3$, we decode the set of represented k-mers $\{\texttt{ACG}, \texttt{GGG}\}$.

3 Function-Assigned Masked Superstrings

Suppose we are given a masked superstring (M, S) and our objective is to determine whether a given k-mer Q is among the represented k-mers. Conceptually, this process consists of two steps: first, identify the occurrences of Q in S and retrieve the corresponding mask symbols; then, verify whether at least one 1 is present. 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.,*

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

In this notation, verifying k-mer presence corresponds to evaluating the composite function '**or** $\circ\, \lambda$'; i.e., k-mer is present if $\lambda(S, M, Q)$ is non-empty and **or** of the values is 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 is OFF and the second ON, and the **or** of these values is 1; therefore, GGG is represented. The set of all represented k-mers for a masked superstring (S, M) is then

$$K = \{Q \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}^k \mid f(\lambda(S, M, Q)) = 1\} \ ,$$

where f is the **or** function.

Nevertheless, **or** is not the only function f that is applicable for such a "demasking", since for instance, with **xor**, we would consider a k-mer present if and only if there is an odd number of ON occurrences of Q; see Table 2 for an example. In fact, k-mer demasking can be done with any Boolean function; see an overview in Table 1. Furthermore, it is convenient to allow the function to reject some input sequences as invalid by returning a special value called invalid, which can also be viewed as restricting mask domain and thus enforcing certain criteria on mask validity. Finally, we limit ourselves to symmetric functions only, as these will later provide useful guarantees for indexing.

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

Function name	Definition	Comprehensive	Use cases
or	0 if $\lvert\lambda\rvert_1 = 0$ 1 if $\lvert\lambda\rvert_1 > 0$ `invalid` never	yes	• The default f-masked superstring (Section 3) • Generalizes (r)SPSS representations (Appendix A) • Union input and output function (Section 4)
xor	0 if $\lvert\lambda\rvert_1$ is even 1 if $\lvert\lambda\rvert_1$ is odd `invalid` never	yes	• Symmetric difference input and output function (Section 4)
and	0 if $\lambda = \epsilon$ or $\lvert\lambda\rvert_0 > 0$ 1 if $\lambda \neq \epsilon$ and $\lvert\lambda\rvert_0 = 0$ `invalid` never	yes	• Allows for ON occurrences of ghost k-mers (Appendix C)
$[a,b]$-**threshold** ($1 \leq a \leq b$)	0 if $\lvert\lambda\rvert_1 < a$ or $\lvert\lambda\rvert_1 > b$ 1 if $a \leq \lvert\lambda\rvert_1 \leq b$ `invalid` never	iff $a = 1$	• Intersection and set difference output function (Section 4)
one-or-nothing	0 if $\lvert\lambda\rvert_1 = 0$ 1 if $\lvert\lambda\rvert_1 = 1$ `invalid` otherwise	yes	• Union, symmetric difference and intersection input function (Section 4) • Set difference left input function (Section 4)
two-or-nothing	0 if $\lvert\lambda\rvert_1 = 0$ 1 if $\lvert\lambda\rvert_1 = 2$ `invalid` otherwise	no	• Set difference right input function (Section 4)
all-or-nothing	0 if $\lvert\lambda\rvert_1 = 0$ 1 if $\lambda \neq \epsilon$ and $\lvert\lambda\rvert_0 = 0$ `invalid` otherwise	yes	• No need for mask rank in queries (Appendix C)

Table 1 Overview of selected demasking functions f for f-masked superstrings. The table includes functions used for set operations as well as in other contexts throughout the paper. In the definitions, we abbreviate $\lambda(f, S, M)$ as λ. Note also that even non-comprehensive functions in this table satisfy properties (P1) and (P4) from Definition 4.

However, not all demasking functions are practically useful, and we will typically require them to have several natural properties. First, we require the non-appearing k-mers to be treated as non-represented, which is ensured by property (P1) in Definition 4 below. (Naturally, if we want to represent the complement of a set, we would treat the non-appearing k-mers as represented.) Second, property (P2) guarantees that for any appearing k-mer Q with any number of occurrences in a given superstring, we can set the mask bits in order to both make Q represented and not represented. Third, even with (P1) and (P2), there is an ambiguity in the meaning of `0` and `1` in the mask and thus, in (P3), we require the `1` to have the meaning of a k-mer being represented; namely, if it has a single occurrence masked with `1`, it should be treated as represented. Finally, in (P4), we require the function to be efficiently computable, specifically in $O(1)$ time from the frequencies of `0`s and `1`s in its input. We call demasking functions satisfying these properties *comprehensive*.

▶ **Definition 4.** *We say that a demasking function f is* comprehensive *if it satisfies the following three properties:*

(P1) $f(\epsilon) = 0$.
(P2) For every $n > 0$, there 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 notion of demasking functions f in hand, we generalize the concept of masked superstrings to so-called f-masked superstrings.

▶ **Definition 5.** *Given a demasking function f, a superstring S, and a binary mask M with $|M| = |S|$, we call a triplet $\mathcal{S} = (f, S, M)$ a* function-assigned masked superstring *or* f-masked superstrings*, abbreviated as* f-MS*.*

Since we allow the output of f to be `invalid`, it may happen that for some f-masked superstring (f, S, M) and some k-mer Q, the result of f for the occurrences of Q turns out to be `invalid`. We call such f-masked superstring *invalid* and will always ensure validity for all masked superstrings that we will work with.

For a valid f-masked superstring, the set of represented k-mers is

$$K = \{Q \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}^k \mid f(\lambda(S, M, Q)) = 1\}.$$

The following observation is a consequence of property (P2) in Definition 4 of comprehensive demasking functions.

▶ **Observation 6.** *For a comprehensive demasking functions f, it holds that for any k-mer set K and any superstring S of k-mers in K, there exists a mask M such that (f, S, M) represents exactly K.*

This further means that for any f-MS (f, S, M) and any comprehensive demasking function g, it is possible to find a mask M' such that (g, S, M') represents exactly the same set as (f, S, M).

▶ **Example 7.** Consider Example 1 with the set of 3-mers $K = \{\mathtt{ACG}, \mathtt{GGG}\}$, a superstring $S = \mathtt{ACGGGG}$, and a mask $M = \mathtt{101100}$. Then the occurrence function for $Q = \mathtt{GGG}$ is $\lambda(S, M, Q) = (1, 1)$. If we choose the f to be **or**, then $f(\lambda(S, M, Q)) = 1$ and thus, `GGG` is represented. However, if we choose **xor** instead, `GGG` is considered as a ghost k-mer.

For **or**, the represented set is $\{\mathtt{ACG}, \mathtt{GGG}\}$ and since **xor** is a comprehensive function, there always exists a mask M' such that $(\mathbf{xor}, S, M')$ represents the same set as $(\mathbf{or}, S, M)$. In our case M' could be `100100`. For functions which are not comprehensive, this is in general impossible. For instance, considering the (impractical) constant-zero function, the represented set will always be empty.

4 f-Masked Superstrings as an Algrebraic Framework

In this section, we describe on the conceptual level how to perform set operations on k-mer sets by simply concatenating f-masked superstrings 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-masked superstrings as concatenating the underlying superstrings and masks for all possible input and output functions f.

▶ **Definition 8.** *Given a function-assigned masked superstring (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 function-assigned masked superstrings 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 8 can be easily extended to more than two input f-masked superstrings. 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 9.** *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*. See Table 2 for an example of function-preserving set operations.

	k-mer set 1	k-mer set 2	set from concatenation
Example f-MSes	$\mathcal{S}_1 = \texttt{AGc}$	$\mathcal{S}_2 = \texttt{CgGCg}$	$\mathcal{S} = \mathcal{S}_1 + \mathcal{S}_2 = \texttt{AGcCgGCg}$
ON occurrences	$1 \times \texttt{AG}$, $1 \times \texttt{GC}$	$1 \times \texttt{GC}$, $2 \times \texttt{CG}$	$1 \times \texttt{AG}$, $2 \times \texttt{GC}$, $2 \times \texttt{CG}$
OFF occurrences	none	$1 \times \texttt{GG}$	$1 \times \texttt{CC}$, $1 \times \texttt{GG}$
f interpreted as **or**			
represented set K	$\{\texttt{AC}, \texttt{GC}\}$	$\{\texttt{CG}, \texttt{GC}\}$	$\{\texttt{AC}, \texttt{CG}, \texttt{GC}\}$
ghost set X	$\{\}$	$\{\texttt{GG}\}$	$\{\texttt{CC}, \texttt{GG}\}$
f interpreted as **xor**			
represented set K	$\{\texttt{AC}, \texttt{GC}\}$	$\{\texttt{GC}\}$	$\{\texttt{AC}\}$
ghost set X	$\{\}$	$\{\texttt{CG}, \texttt{GG}\}$	$\{\texttt{CC}, \texttt{GC}, \texttt{CG}, \texttt{GG}\}$

Table 2 Represented 2-mer sets with or and xor for masked superstrings and their concatenation. The second and third row depict the ON and OFF occurrences of 2-mers, respectively, in the uni-directional model for masked superstrings $\mathcal{S}_1 = \texttt{AGc}$, $\mathcal{S}_2 = \texttt{CgGCg}$ and their concatenation. Note that after the concatenation new OFF occurrences of boundary k-mers emerge; in this case, it is just the blue-colored 2-mer CC. The bottom part depicts the represented sets K and ghost sets X for these f-masked superstring when interpreted using **or** and **xor**. Note that $K_{\mathbf{or}} = K_1 \cup K_2$ in the case of **or** and $K_{\mathbf{xor}} = K_1 \Delta K_2$ for **xor**.

4.2 Function-Preserving Set Operations

Union. As implicitly shown in [61], concatenating masked superstrings, which are **or**-masked superstrings 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**-masked superstrings to generalize (r)SPSS representations, since any set of k-mers in the (r)SPSS representation can be directly viewed as an **or**-masked superstring 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**-masked superstrings are the only f-masked superstrings 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**-masked superstring 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 masked superstrings, 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-masked superstrings are denoted $[a,b]$-**threshold**-masked superstrings.
- 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**-masked superstring 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🐫 [61].

We then concatenate the individual **one-or-nothing**-masked superstring. The result is not a valid **one-or-nothing**-masked superstring 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-masked superstring from **one-or-nothing** to an $[a,b]$-**threshold** function. This will result in an $[a,b]$-**threshold**-masked superstring 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**-masked superstrings in this setting include the following:

- The $[N,N]$-**threshold**-masked superstring corresponds to taking the intersection of the represented sets.
- The $[1,N]$-**threshold**-masked superstring is the **or**-masked superstring and corresponds to taking the union.
- The $[1,1]$-**threshold**-masked superstring 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**-masked superstrings in an arbitrary order. The symmetry of **op** implies that there is a set $S_N \subseteq \{0, 1, \dots, N\}$ 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 $N = 1, 2, \dots$ 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.

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 $(1, 1)$-masked superstring,
- represent B using a $(2, 2)$-masked superstring, and
- interpret the result as a $(1, 1)$-masked superstring.

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-masked superstrings, although they may not correspond to set operations. In Appendix C, we mention the **and** and **all-or-nothing** demasking functions that could be useful for some applications (see also Table 1).

5 f-Masked Superstrings as a Data Type

After seeing f-masked superstrings as an algebraic framework in Section 4, here we demonstrate how to turn them into a standalone data type for k-mer sets supporting all the key operations. This consists of using the FMS-index, an adaptation of the FM-index [24], as an underlying support structure for indexing, and using its capabilities for implementing all the operations. One specific prototype implementation is then described and evaluated in Section 6.

5.1 FMS-Index: An FM-Index Tailored to f-Masked Superstrings

As f-masked superstrings form a textual representation k-mer sets, it is natural to index them using full-text indexes (see, e.g., review in [48]). Given its intrinsic properties and the availability of high-quality implementations, the most natural choice is the FM-index [24]. However, due to the presence of the mask, applying the FM-index directly does not work.

We thus introduce the FMS-index, a modified version of the FM-index, which omits sampled suffix arrays and adds an auxiliary table for the mask. The key idea is to store the BWT image of the superstring and transformed mask such that the i-th symbol of the transformed mask corresponds to the mask symbol at the starting position of the i-th lexicographically smallest suffix of the superstring. Since the transformed mask can be computed alongside the construction of the superstring FM-index, the mask transformation requires only linear time.

In the remainder of this section, we first describe operations such as exporting, merging, mask recasting, and compaction, which are used as building blocks of more high-level operations. Then we focus on answering membership queries on f-masked superstrings and how to efficiently perform the set operations as conceptually described in Section 4 using merging of FM-indexes.

5.2 Basic Operations with f-Masked Superstrings

To implement operations for f-masked superstrings 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 a f-MS representation 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-masked superstrings, concatenate them in their string form and index the result, which achives linear time complexity. Alternatively, it is possible merge the indexes without exporting by directly merging the BWTs [28] 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 [61]. 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-MSes, 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-masked superstring, counting the number of ON and OFF occurrences of each k-mer, constructing the represented k-mer set, and computing its **or**-MS [61]. Alternatively, one may directly compute an f-masked

superstring using the local greedy algorithm [61] executed on the FMS-index, as described in Appendix D for the uni-directional model.

5.3 Membership Queries via FMS-Index

The process of answering membership queries on indexed f-masked superstrings can be split into two steps. First, we use the FMS-index of the superstring to find the range in the suffix-array coordinates corresponding to the queried k-mer and also the range for its reverse complement. Note that unlike in the FM-index, we do not have to translate each occurrence to the original coordinates.

Second, we determine the presence of the k-mer using the transformed mask. Since demasking functions are by Definition 3 symmetric, in order to evaluate them, we do not need to know the exact order of ON and OFF occurrences as the respective frequencies are sufficient. We compute these frequencies by rank queries on the transformed mask in the corresponding range as the occurrences of λ for a particular k-mer are located next to each other. Given the number of ON and OFF occurrences (in total for the queried k-mer and its reverse complement), we evaluate the demasking function to answer the membership query.

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 4, the time complexity is constant.

5.4 Performing Set Operations on Indexed k-Mer Sets

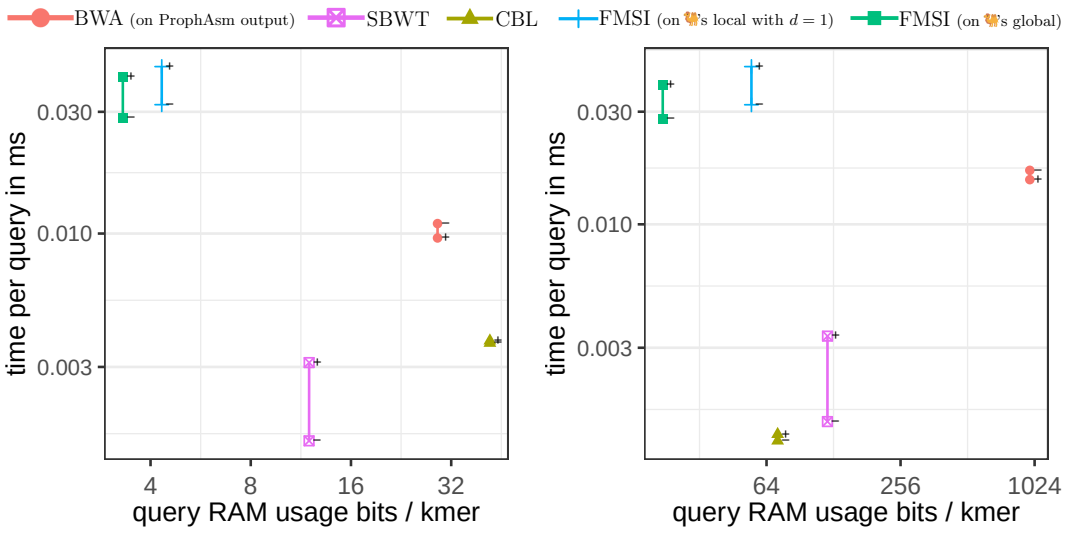
Using an indexed f-MS, set operations such as union, (a)symmetric difference, or intersection can be performed directly via their associated abstract operations in Section 4. Indeed, we implement concatenation of masked superstrings via index merging. Prior to concatenating, we only need to ensure that 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-masked superstring 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.

6 Experimental Evaluation

Implementation of f-masked superstring operations in FMSI. We implemented indexing f-masked superstring and the associated k-mer set operations in a tool called FMSI (f-Masked Superstring Index). The tool supports membership queries on indexed f-masked superstrings 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-MSes, and then reindexing. Compaction is implemented in two variants: first, using k-mer counting and KmerCamel🐫 [61] to construct a superstring; second, using a version of the local algorithm on FMS-index described in Appendix D to work in the bi-directional model at the cost of having $O(k)$-times higher time complexity. All the demasking functions mentioned in Table 1 are supported directly by FMSI, and users can possibly add their custom ones.



(a) $k = 23$, without subsampling k-mers. **(b)** $k = 23$, subsampled k-mers at rate 0.1.

Figure 1 Query time and memory for the *E. coli* pan-genome. For each of the algorithms, we plot two points, one for positive queries (+) and one for negative (-), connected by a line.

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 [25], 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 `a36c2e3`) 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**[1] [38], a state-of-the-art aligner based on the FM index; for processing queries, we used the `fastmap` command [36], run with parameter $w = 999999$ on the simplitigs computed by ProphAsm [11],
- **SBWT**[2] [1], 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] and added all reverse complements to the index, and

[1] `https://github.com/lh3/bwa`, commit `139f68f`.
[2] `https://github.com/algbio/SBWT`, commit `c433b53`.

- **CBL**[3] [45], a very recent method based on smallest cyclic rotations of k-mers.

We have run FMSI on the masked superstrings computed by KmerCamel🐫 [61], 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 [5]) and *S. pneumoniae* pan-genome (computed from 616 assemblies from a study of children in Massachusetts, USA [21]) 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.

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; for further results, we refer to the supplementary repository[4]. Across all of the datasets, values of k, and subsampling rates, FMSI run on the masked superstring computed by KmerCamel🐫's global greedy required 3-10 times less memory for processing queries than all of the other methods, attaining around 3-4 bits per k-mer on non-subsampled *E. coli* pan-genome. However, FMSI needed substantially more time for processing queries than the other programs; this is mainly due to the prototype nature of our implementation, and we believe that the query time of FMSI can be substantially optimized. SBWT (in the plain-matrix variant) and CBL are generally the fastest algorithms for processing queries. We note that SBWT required substantial disk space during index construction (up to tens of GBs for the *E. coli* pan-genome). We also remark that while the memory usage per k-mer grows with decreasing subsampling rate, the query time remains roughly the same for all algorithms.

Experimental evaluation of set operations. We demonstate the feasibility of using FMSI to perform set operations on k-mer sets. 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**-masked superstrings. 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-masked superstrings 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-masked superstring, and the memory requirements to perform queries on the indexed individual and concatenated f-masked superstrings, both before and after compaction. The results for intersection of the two roundworm genomes for $k = 23$ are depicted in Figure 2. Overall, at every step, the memory required to query the indexed f-masked superstring 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

[3] `https://github.com/imartayan/CBL`, commit `8e8f28e`.
[4] `https://github.com/OndrejSladky/f-masked-superstrings-supplement`

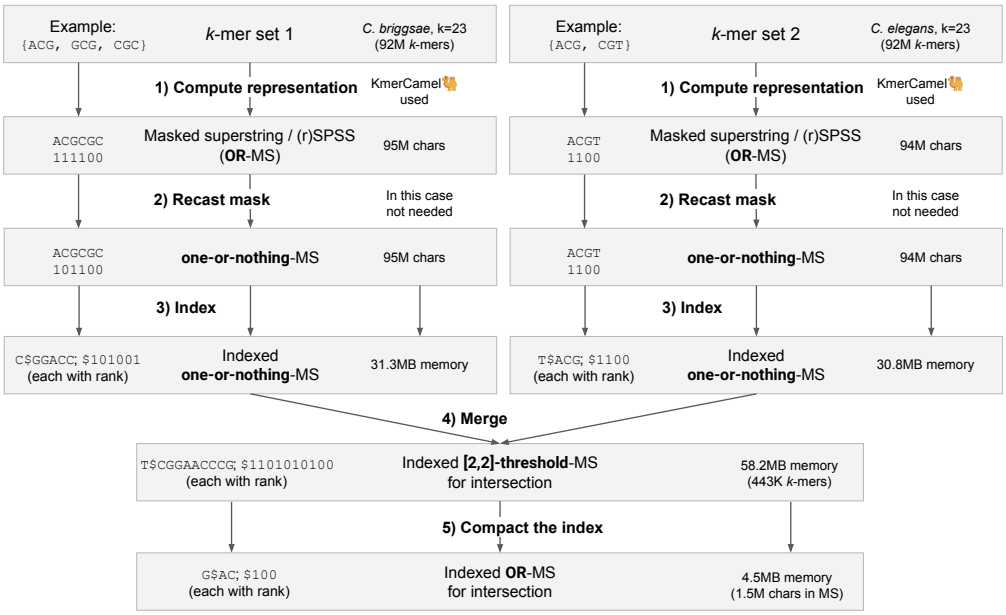


Figure 2 Set operations workflow using f-masked superstrings: example on intersection. The workflow also contains illustrative example on a set of 3-mers as well as time for the operations and experimental data for running the workflow on *C. briggsae* and *C. elegans* genomes with $k = 23$. The experimental 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.

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 reduces the masked superstring 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[4].

7 Discussion and Conclusions

We have proposed f-masked superstrings 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 from [61] 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 masked superstrings 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 masked superstrings 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-masked superstrings 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 still lacks behind state-of-the-art methods in terms of query time. However, we believe that FMSI can be substantially optimized; this is supported by the fact that both FMSI and BWA [38, 36] are based on the FM index and therefore, it should be possible to reduce query times of FMSI on the level of BWA. We also note that we did not optimize other subroutines of FMSI, such as index merging. Furthermore, we leave as an open question how to design efficient algorithms for mask recasting in the suffix-array coordinates and compaction in the bi-directional model without the need to export the f-masked superstring.

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 querying consecutive k-mers of a genomic sequence (e.g., a read obtained from sequencing) can be added by using the bi-directional FM-index [33] (details left to future work). Another significant direction is to extend f-masked superstrings for indexing large collections of k-mer sets (see, e.g., a review in [42]). 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.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. Grace A. Blackwell, Martin Hunt, Kerri M. Malone, Leandro Lima, Gal Horesh, Blaise T. F. Alako, Nicholas R. Thomson, and Zamin Iqbal. Exploring bacterial diversity via a curated and searchable snapshot of archived dna sequences. *PLOS Biology*, 19(11):1–16, 11 2021. doi:10.1371/journal.pbio.3001421.
6. 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.

7 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.

8 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.

9 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.

10 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.

11 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.

12 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.

13 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. doi:10.1101/2023.04.15.536996.

14 Karel Břinda, Kamil Salikhov, Simone Pignotti, and Gregory Kucherov. Prophyle 0.3.1.0. *Zenodo*, 5281, 2017. doi:10.5281/zenodo.5237391.

15 Rayan Chikhi. K-mer data structures in sequence bioinformatics. HDR thesis, Institut Pasteur Ecole Doctorale "EDITE", 2021.

16 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.

17 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.

18 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.

19 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.

20 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.

21 Nicholas J. Croucher, Jonathan A. Finkelstein, Stephen I. Pelton, Julian Parkhill, Stephen D. Bentley, Marc Lipsitch, and William P. Hanage. Population genomic datasets describing the post-vaccine evolutionary epidemiology of streptococcus pneumoniae. *Scientific Data*, 2, 2015. doi:10.1038/sdata.2015.58.

22 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.

23 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.

24 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

25 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

26 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.

27 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.

28 James Holt and Leonard McMillan. Merging of multi-string bwts with applications. *Bioinformatics*, 30(24):3524–3531, August 2014. URL: http://dx.doi.org/10.1093/bioinformatics/btu584, doi:10.1093/bioinformatics/btu584.

29 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.

30 Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Christopher Barber, Marc Zimmermann, Gunnar Rätsch, and André Kahles. Metagraph: Indexing and analysing nucleotide archives at petabase-scale. *BioRxiv*, pages 2020–10, 2020. doi:10.1101/2020.10.01.322164.

31 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. *Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications*, pages 181–192. Springer Berlin Heidelberg, 2001. doi:10.1007/3-540-48194-x_17.

32 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.

33 T. W. Lam, Ruiqiang Li, Alan Tam, Simon Wong, Edward Wu, and S. M. Yiu. High throughput short read alignment via bi-directional bwt. In *2009 IEEE International Conference on Bioinformatics and Biomedicine*. IEEE, 2009. doi:10.1109/bibm.2009.42.

34 John A Lees, Marco Galardini, Stephen D Bentley, Jeffrey N Weiser, and Jukka Corander. pyseer: a comprehensive tool for microbial pangenome-wide association studies. *Bioinformatics*, 34(24):4310–4312, 2018. doi:10.1093/bioinformatics/bty539.

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 Po-Ru Loh, Michael Baym, and Bonnie Berger. Compressive genomics. *Nature Biotechnology*, 30(7):627–630, July 2012. URL: http://dx.doi.org/10.1038/nbt.2241, doi:10.1038/nbt.2241.

42 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.

43 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.

44 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.

45 Igor Martayan, Bastien Cazaux, Antoine Limasset, and Camille Marchet. Conway-Bromage-Lyndon (CBL): an exact, dynamic representation of k-mer sets. *bioRxiv*, 2024. doi:10.1101/2024.01.29.577700.

46 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.

47 Martin D. Muggli, Alexander Bowe, Noelle R. Noyes, Paul S. Morley, Keith E. Belk, Robert Raymond, Travis Gagie, Simon J. Puglisi, and Christina Boucher. Succinct colored de Bruijn graphs. *Bioinformatics*, 33(20):3181–3187, 2017. doi:10.1093/bioinformatics/btx067.

48 Gonzalo Navarro. *Texts*, page 395–449. Cambridge University Press, 2016.

49 Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. Squeakr: an exact and approximate k-mer counting system. *Bioinformormatics*, 34(4):568–575, 2018. doi:10.1093/bioinformatics/btx636.

50 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.

51 Giulio Ermanno Pibiri. Sparse and skew hashing of K-mers. *Bioinformatics*, 38(Supplement_1):i185–i194, 2022. doi:10.1093/bioinformatics/btac245.

52 Amatur Rahman. Compression algorithms for de Bruijn graphs and uncovering hidden assembly artifacts. PhD thesis, The Pennsylvania State University, 2023.

53 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.

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. 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 Liam P Shaw, Eduardo P C Rocha, and R Craig MacLean. Restriction-modification systems have shaped the evolution and distribution of plasmids across bacteria. *Nucleic Acids Research*, 51(13):6806–6818, 2023. doi:10.1093/nar/gkad452.

60 Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Efficient Reconciliation of Genomic Datasets of High Similarity. In Christina Boucher and Sven Rahmann, editors, *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*, volume 242 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.WABI.2022.14.

61 Ondřej Sladký, Pavel Veselý, and Karel Břinda. Masked superstrings as a unified framework for textual k-mer set representations. *bioRxiv*, 2023. doi:10.1101/2023.02.01.526717.

62 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.

63 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 [61], which we call **or**-masked superstrings, have the important property that concatenating them results in the union of represented k-mers. This property makes it possible for masked superstring to generalize (r)SPSS representations [61] as unifying individual simplitigs/matchtigs results in correctly representing the union of respective represented k-mer sets. In this section, we show that **or**-masked superstrings are the only f-masked superstrings with any of these properties (acting as union and generalizing (r)SPSS).

▶ **Theorem 10.** ***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-masked superstrings (f, S, M) and (f, S', M'), respectively, their concatenation $(f, S{+}S', M{+}M')$ is a valid f-masked superstring representing the set $K \cup K'$.*

Proof of Theorem 10. 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-masked superstring 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-masked superstring 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-masked superstring is either invalid or does not contain the k-mer in the represented set as $f(x) \neq 1$, a contradiction. ◀

▶ **Theorem 11.** ***or*** *is the only comprehensive demasking function f such that for any sequence of f-masked superstrings, where individual superstrings are matchtigs, the concatenation of all the f-masked superstrings represents the union of represented k-mers.*

Proof of Theorem 11. 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 10.

We do this with k-mer `CG` and matchtigs `Cg` and `Gc`. Consider the counterexample sequence of occurring ones and zeros from Theorem 10. 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 10. ◀

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-masked superstrings 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-masked superstrings. 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 masked superstrings 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 12.** *There is no comprehensive demasking function f with the property that the result of f-concatenation of two f-masked superstrings always represents the intersection of the originally represented k-mer sets.*

Proof of Theorem 12. Let f be any comprehensive demasking function. Consider masked superstrings `A` and `C`, each representing a single 1-mer. Their concatenation is `AC`. Since $f_1(1) = 1$ 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-masked superstrings with comprehensive functions solely.

▶ **Theorem 13.** *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 sketch. Consider a k-mer Q appearing exactly once in both of the input superstrings such that it is represented in one of the input sets and does not have a boundary occurrence after concatenating the superstrings. By the symmetry of the demasking function f_o, we get the same result if Q appears in the first set, implying that it should be represented in the result, as if Q appears in the second set, in which case it should be treated as ghost in the result. Hence, the function f_o cannot be correctly representing the difference. ◀

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**-masked superstrings easier than indexing general f-masked superstrings, 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 5.3.

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**-masked superstrings can be viewed as or-masked superstrings that maximize the number of ones in the mask.

C.2 The and-masked superstrings

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**-masked superstrings over **or**-masked superstrings 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 Local Greedy Using FMS-Index

In this section, we present an implementation of the local greedy algorithm [61] for masked superstring computation in the uni-directional model (i.e., without considering a k-mer and its reverse complement as equivalent) that uses FMS-index of Section 5.1. More precisely, we require a bi-directional version of FMS-index, based on the bi-directional FM-index [33], as the underlying data structure alongside with the transformed mask.

Additionally to the bi-directional FMS-index and the mask, we require a $kLCP_{0-1}$ [54] data structure, that for each position determines whether the longest common prefix of the two neighbouring suffixes is of length at least k. The $kLCP_{0-1}$ array can be computed trivially from the LCP array which, for an input string S, can be computed in $O(|S|)$ time during the construction of FM-index [31].

The local greedy algorithm with parameter d_{max} proceeds as follows. It first chooses an arbitrary k-mer that has not been represented yet. Then it tries to extend it to both sides via extensions of length d starting with $d = 1$ and going up to $d = d_{max}$; see Appendix C of [61] for more details. Regarding the implementation of the local greedy, there are two issues to address: First, how to maintain the k-mers that have not been represented yet, and second, how to quickly check whether a k-mer exists.

We start with the lexicographically smallest k-mer, which is the one that appears first in the suffix coordinates, and based on the $kLCP_{0-1}$ array, we find the last occurrence of this k-mer. From the mask, we determine whether the k-mer is represented. If not we continue to the next k-mer, and otherwise, we delete it. This can be done by finding the number of 1s such that f evaluates to zero and then setting the mask symbols in the range to that many 1s.

Then we try to extend the selected k-mer in both directions. Adding the extension characters to either direction can be done directly using the bi-directional FM-index and removing the characters in order to keep the string a k-mer can be done using the $kLCP_{0-1}$ array [54]. We check whether this k-mer is represented, if so delete the k-mer, extend the string, and continue.

Apart from extending the current string, which takes $O(4^{d_{max}})$ per extension, each position is visited at most once per deletion and once when scanning for initial k-mers. Thus, the time complexity of the algorithm is $O(|S| + N4^{d_{max}})$ where N is the number of represented k-mers, which is linear for constant values of d_{max}.

To implement this approach in the bi-directional model (where a k-mer is equivalent to its reverse complement), we would need to locate the reverse complement of each k-mer, which would worsen the time complexity by a factor of k. We leave it as an open question whether the same time complexity as in the uni-directional model can be obtained in the bi-directional model as well. However, we note that for practical usage, the uni-directional algorithm is usable also in the bi-directional model as only the canonical k-mer from the pair (i.e., the lexicographically smaller one) can be stored and queried.