



HAL
open science

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

Ondřej Sladký, Pavel Veselý, Karel Břinda

► **To cite this version:**

Ondřej Sladký, Pavel Veselý, Karel Břinda. Function-Assigned Masked Superstrings as a Versatile and Compact Data Type for k -Mer Sets. 2024. hal-04573444

HAL Id: hal-04573444

<https://hal.science/hal-04573444>

Preprint submitted on 13 May 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

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

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

87 1.1 Related Work

88 A large body of work focused on data structures for k -mer sets and their collections; we
 89 refer to [16, 42] for recent surveys. One approach for individual k -mer sets is to combine
 90 the aforementioned textual k -mer set representations based on the de Bruijn graph of the
 91 set [17, 11, 53, 58], i.e., (r)SPSS, with efficient string indexes such as the FM index [24] or
 92 BWA [38, 39, 36, 37] (see also ProPhex [55, 54]). Another approach based on BWT is the
 93 BOSS data structure [6] that was implemented as an index for a collection of k -mer sets in
 94 VARI [47], VARI-merge [46], and Metagraph [30]. Taking BOSS as an inspiration, Spectral
 95 Burrows-Wheeler Transform (SBWT) [1] has been proposed as a compact representation of
 96 the de Bruijn graph, together with various approaches for its indexing. Themisto [2] further
 97 builds on SBWT and provides an index for collections of k -mer sets.

98 Hashing techniques have also been successful in the design of k -mer data structures. In
 99 particular, Bifrost [27] uses hash tables for indexing colored de Bruijn graphs, which encode
 100 collections of k -mer sets. Other popular data structures, such as BBHash [40], BLight [44],
 101 and SSHash [51], employ minimal perfect hash functions and serve as a base for constructing
 102 indexes such as FDBG [20], REINDEER [43], Fulgor [23], and pufferfish2 [22].

103 While the aforementioned approaches yield exact data structures, further space com-
 104 pression may be achieved by employing probabilistic techniques and allowing for a certain
 105 low-probability error, such as false positives. Namely, the counting quotient filter was used
 106 for k -mers in Squeakr [49], and this data structure was extended to an efficient index called
 107 dynamic Mantis [3]. Another line of work, e.g., [7, 4, 26, 35], employs variants of the
 108 Bloom filter to further reduce space requirements. Recently, Invertible Bloom Lookup Tables
 109 combined with subsampling have been used to estimate the Jaccard similarity coefficient [60].

110 Very recently, the Conway-Bromage-Lyndon (CBL) structure [45] builds on the work
 111 of Conway and Bromage [19] and combines smallest cyclic rotations of k -mers with sparse
 112 bit-vector encodings, to yield a dynamic and exact k -mer index supporting set operations,
 113 such as union, intersection, and difference. Finally, we note that set operations can also be
 114 carried out using some k -mer lists and counters, e.g., [29, 32]; however, these methods are
 115 unable to exploit structural properties of k -mer sets such as the SLP.

116 1.2 Our Contribution and Organization of the Paper

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

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

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

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

148 2 Preliminaries

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

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

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

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

23:4 f -Masked Superstrings as a Data Type

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

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

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

186 3 Function-Assigned Masked Superstrings

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

192 ► **Definition 2.** For a superstring S , a mask M , and a k -mer Q , the occurrence function
193 $\lambda(S, M, Q) \rightarrow \{0, 1\}^*$ is a function returning a finite binary sequence with the mask symbols
194 of the corresponding occurrences, i.e.,

$$195 \quad \lambda(S, M, Q) := (M_i \mid S_i \dots S_{i+k-1} = Q) .$$

196 In this notation, verifying k -mer presence corresponds to evaluating the composite function
197 ‘`or` \circ λ ’; i.e., k -mer is present if $\lambda(S, M, Q)$ is non-empty and `or` of the values is 1. For
198 instance, in Example 1 for the k -mer $Q = \text{GGG}$, it holds that $\lambda(S, M, Q) = (0, 1)$, as the
199 first occurrence is OFF and the second ON, and the `or` of these values is 1; therefore, `GGG`
200 is represented. The set of all represented k -mers for a masked superstring (S, M) is then

$$201 \quad K = \{Q \in \{\text{A}, \text{C}, \text{G}, \text{T}\}^k \mid f(\lambda(S, M, Q)) = 1\} ,$$

202 where f is the `or` function.

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

211 ► **Definition 3.** We call a symmetric function $f : \{0, 1\}^* \rightarrow \{0, 1, \text{invalid}\}$ a k -mer
212 demasking function.

Function name	Definition	Comprehensive	Use cases
or	0 if $ \lambda _1 = 0$ 1 if $ \lambda _1 > 0$ invalid never	yes	<ul style="list-style-type: none"> • The default f-masked superstring (Section 3) • Generalizes (r)SPSS representations (Appendix A) • Union input and output function (Section 4)
xor	0 if $ \lambda _1$ is even 1 if $ \lambda _1$ is odd invalid never	yes	<ul style="list-style-type: none"> • Symmetric difference input and output function (Section 4)
and	0 if $\lambda = \epsilon$ or $ \lambda _0 > 0$ 1 if $\lambda \neq \epsilon$ and $ \lambda _0 = 0$ invalid never	yes	<ul style="list-style-type: none"> • Allows for ON occurrences of ghost k-mers (Appendix C)
$[a,b]$-threshold ($1 \leq a \leq b$)	0 if $ \lambda _1 < a$ or $ \lambda _1 > b$ 1 if $a \leq \lambda _1 \leq b$ invalid never	iff $a = 1$	<ul style="list-style-type: none"> • Intersection and set difference output function (Section 4)
one-or-nothing	0 if $ \lambda _1 = 0$ 1 if $ \lambda _1 = 1$ invalid otherwise	yes	<ul style="list-style-type: none"> • Union, symmetric difference and intersection input function (Section 4) • Set difference left input function (Section 4)
two-or-nothing	0 if $ \lambda _1 = 0$ 1 if $ \lambda _1 = 2$ invalid otherwise	no	<ul style="list-style-type: none"> • Set difference right input function (Section 4)
all-or-nothing	0 if $ \lambda _1 = 0$ 1 if $\lambda \neq \epsilon$ and $ \lambda _0 = 0$ invalid otherwise	yes	<ul style="list-style-type: none"> • 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.

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

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

227 (P1) $f(\epsilon) = 0$.

228 (P2) For every $n > 0$, there exist $x, y \in \{0, 1\}^n$ such that $f(x) = 0$ and $f(y) = 1$.

229 (P3) $f((1)) = 1$ and $f((0)) = 0$.

230 (P4) Given $|x|_0$ and $|x|_1$, one can evaluate $f(x)$ in constant time in the wordRAM model.

231 With the notion of demasking functions f in hand, we generalize the concept of masked
232 superstrings to so-called f -masked superstrings.

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

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

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

$$241 \quad K = \{Q \in \{\mathbf{A}, \mathbf{C}, \mathbf{G}, \mathbf{T}\}^k \mid f(\lambda(S, M, Q)) = 1\}.$$

242 The following observation is a consequence of property (P2) in Definition 4 of comprehen-
243 sive demasking functions.

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

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

250 ► **Example 7.** Consider Example 1 with the set of 3-mers $K = \{\mathbf{ACG}, \mathbf{GGG}\}$, a superstring
251 $S = \mathbf{ACGGGG}$, and a mask $M = \mathbf{101100}$. Then the occurrence function for $Q = \mathbf{GGG}$ is
252 $\lambda(S, M, Q) = (1, 1)$. If we choose the f to be `or`, then $f(\lambda(S, M, Q)) = 1$ and thus, \mathbf{GGG} is
253 represented. However, if we choose `xor` instead, \mathbf{GGG} is considered as a ghost k -mer.

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

259 **4 f -Masked Superstrings as an Algebraic Framework**

260 In this section, we describe on the conceptual level how to perform set operations on k -
261 mer sets by simply concatenating f -masked superstrings and choosing suitable demasking
262 functions f . We deal with practical aspects of efficient implementation of this concept in
263 Section 5.

264 **4.1 Concatenation as an Elementary Low-Level Operation**

265 We define concatenation on f -masked superstrings as concatenating the underlying super-
266 strings and masks for all possible input and output functions f .

267 ► **Definition 8.** Given a function-assigned masked superstring (f_1, S_1, M_1) and (f_2, S_2, M_2) ,
268 we define (f_1, f_2, f_o) -concatenation as the operation taking these two function-assigned masked
269 superstrings and producing the result $(f_o, S_1 + S_2, M_1 + M_2)$. We denote this operation by
270 $+_{f_1, f_2, f_o}$.

271 Note that Definition 8 can be easily extended to more than two input f -masked superstrings.
 272 In the case that all the functions are the same, i.e. $f = f_1 = f_2 = f_o$, we call it f -concatenation
 273 or just *concatenation* if f is obvious from the context.

274 ► **Definition 9.** We call the set operations that can be performed with $f_1 = f_2 = f_o$ function-
 275 preserving set operations. The operations that cannot be performed with a single function
 276 are called function-transforming set operations.

277 Furthermore, note that while the set of appearing k -mers of $S_1 + S_2$ clearly contains
 278 the union of appearing k -mers of S_1 and of S_2 , additional new occurrences of k -mers may
 279 appear at the boundary of the two superstrings. These newly appearing k -mers may not be
 280 appearing in any of the superstrings S_1 and S_2 . We refer to them as *boundary k -mers* and
 281 to the occurrences of appearing k -mers of $S_1 + S_2$ that overlap both input superstrings as
 282 *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	$S_1 = \text{AGc}$	$S_2 = \text{CgGCg}$	$\mathcal{S} = S_1 + S_2 = \text{AGcCgGCg}$
ON occurrences	$1 \times \text{AG}, 1 \times \text{GC}$	$1 \times \text{GC}, 2 \times \text{CG}$	$1 \times \text{AG}, 2 \times \text{GC}, 2 \times \text{CG}$
OFF occurrences	none	$1 \times \text{GG}$	$1 \times \text{CC}, 1 \times \text{GG}$
f interpreted as or			
represented set K	{AC, GC}	{CG, GC}	{AC, CG, GC}
ghost set X	{}	{GG}	{CC, GG}
f interpreted as xor			
represented set K	{AC, GC}	{GC}	{AC}
ghost set X	{}	{CG, GG}	{CC, GC, CG, 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 $S_1 = \text{AGc}$, $S_2 = \text{CgGCg}$ 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 $K_{\text{or}} = K_1 \cup K_2$ in the case of **or** and $K_{\text{xor}} = K_1 \Delta K_2$ for **xor**.

283 4.2 Function-Preserving Set Operations

284 **Union.** As implicitly shown in [61], concatenating masked superstrings, which are **or**-
 285 masked superstrings in our notation, acts as union on the represented sets, that is, the
 286 resulting represented set is the union of the original represented sets. This allows **or**-masked
 287 superstrings to generalize (r)SPSS representations, since any set of k -mers in the (r)SPSS
 288 representation can be directly viewed as an **or**-masked superstring by concatenating the
 289 individual simplitigs/matchtigs.

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

294 **Symmetric difference.** Next, we observe that **xor** naturally acts as the symmetric
 295 difference set operation, i.e., concatenating two **xor**-masked superstring results in a **xor**-MS
 296 representing the symmetric difference of the original sets. Indeed, recall that using **xor**
 297 implies that a k -mer is represented if and only if there is a odd number of ON occurrences of
 298 that k -mer. Observe that the boundary occurrences of k -mers do not affect the resulting

299 represented set as those have zeros in the mask. Thus, if a k -mer is present in both sets, it
 300 has an even number of ON occurrences in total and hence, is not represented in the result.
 301 Likewise, if a k -mer belongs to exactly one input set, it has an odd number of ON occurrences
 302 in this input set and an even number (possibly zero) in the other; thus, it is represented
 303 in the result. As any appearing k -mer is either boundary or appears in one of the masked
 304 superstrings, the result corresponds to the symmetric difference.

305 4.3 Function-Transforming Set Operations

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

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

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

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

323 We now use these functions to perform any symmetric set operation on any number
 324 of input k -mer sets. Given N sets of k -mers, we compute a **one-or-nothing**-masked
 325 superstring for each. This is always possible since **one-or-nothing** is a comprehensive
 326 demasking function and can be done by directly using the superstrings and masks computed
 327 by KmerCamel 🐫 [61].

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

- 336 ■ The **$[N,N]$ -threshold**-masked superstring corresponds to taking the intersection of the
 337 represented sets.
- 338 ■ The **$[1,N]$ -threshold**-masked superstring is the **or**-masked superstring and corresponds
 339 to taking the union.
- 340 ■ The **$[1,1]$ -threshold**-masked superstring corresponds to taking those k -mers that appear
 341 in exactly one of the original sets. In case of $N = 2$, this corresponds to the symmetric
 342 difference.

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

347 **Arbitrary symmetric set operations.** The same scheme, with more general demasking
 348 functions, can be used to implement *any symmetric set operation \mathbf{op} on any number of*
 349 *sets*. Indeed, given N , we again concatenate their **one-or-nothing**-masked superstrings
 350 in an arbitrary order. The symmetry of \mathbf{op} implies that there is a set $S_N \subseteq \{0, 1, \dots, N\}$
 351 such that a k -mer belongs to the set resulting from applying \mathbf{op} if and only if it is in a
 352 input sets for some $a \in S$. The sets S_N for $N = 1, 2, \dots$ can be directly transferred into a
 353 demasking function $f_{\mathbf{op}}$ that models \mathbf{op} ; however, $f_{\mathbf{op}}$ may not satisfy the property (P4)
 354 from Definition 4.

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

- 360 ■ represent A using a $(1, 1)$ -masked superstring,
- 361 ■ represent B using a $(2, 2)$ -masked superstring, and
- 362 ■ interpret the result as a $(1, 1)$ -masked superstring.

363 This computes the difference correctly as all k -mers represented in B are treated as ghosts
 364 in the result, the k -mers from A but not from B still have a single ON occurrence and thus
 365 are correctly considered represented, and finally, the ghost k -mers in either of the initial sets
 366 or the boundary k -mers have no influence on the result. The same functions can be used if
 367 we subtract more than a single set. Furthermore, this scheme can be generalized to *any set*
 368 *operations on any number of sets*, by representing the i -th input set with (i, i) -MS and using
 369 a suitable demasking function for the result of the concatenation (constructed similarly as
 370 $f_{\mathbf{op}}$ for symmetric operation \mathbf{op} above).

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

376 **Other applications.** Furthermore, there are many more demasking functions that can be
 377 used with f -masked superstrings, although they may not correspond to set operations. In
 378 Appendix C, we mention the **and** and **all-or-nothing** demasking functions that could be
 379 useful for some applications (see also Table 1).

380 **5 f -Masked Superstrings as a Data Type**

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

387 **5.1 FMS-Index: An FM-Index Tailored to *f*-Masked Superstrings**

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

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

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

404 **5.2 Basic Operations with *f*-Masked Superstrings**

405 To implement operations for *f*-masked superstrings using this index, we need to decompose
 406 individual *f*-MS operations into primitive operations with the index. On a low level, these
 407 involve exporting the *f*-MS to its string representation from the index, *f*-MS concatenation,
 408 changing the demasking function, and compaction to decrease its size by data deduplication.

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

413 ***f*-MS concatenation by index merging.** We can export the *f*-masked superstrings,
 414 concatenate them in their string form and index the result, which achieves linear time
 415 complexity. Alternatively, it is possible merge the indexes without exporting by directly
 416 merging the BWTs [28] and masks alongside it.

417 ***f*-MS mask recasting for *f* transformation.** To change the demasking function *f* to
 418 a different one without altering the represented *k*-mer set and the underlying superstring,
 419 we may need to *recast* the mask. Although the recasting procedure depends on the specific
 420 function *f* used, for all comprehensive functions mentioned in Table 1, recasting can be
 421 done either by maximizing the number of 1s in the mask (**and** and **all-or-nothing**), or by
 422 minimizing the number of 1s (all other functions in Table 1). If an *f*-MS is represented in
 423 the original string form, this can be achieved in linear time using a single/two-pass algorithms,
 424 respectively [61]. For *f*-MS in the suffix-array coordinates, we can export the *f*-MS, then
 425 recast the mask, and index the result.

426 ***f*-MS compaction.** If an *f*-MS contains too many redundant copies of individual *k*-mers,
 427 e.g., if an *f*-MS is obtained by concatenating multiple input *f*-MSes, it might be desirable to
 428 *compact* it, i.e., reoptimize its support superstring. This can be performed in linear time,
 429 using two different approaches: One option is exporting the *f*-masked superstring, counting
 430 the number of ON and OFF occurrences of each *k*-mer, constructing the represented *k*-mer
 431 set, and computing its **or**-MS [61]. Alternatively, one may directly compute an *f*-masked

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

434 5.3 Membership Queries via FMS-Index

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

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

447 The first step can be performed in $O(k)$ time. The complexity of the second step depends
448 on how efficiently we can evaluate f . For functions satisfying the property (P4) from
449 Definition 4, the time complexity is constant.


450 5.4 Performing Set Operations on Indexed k -Mer Sets

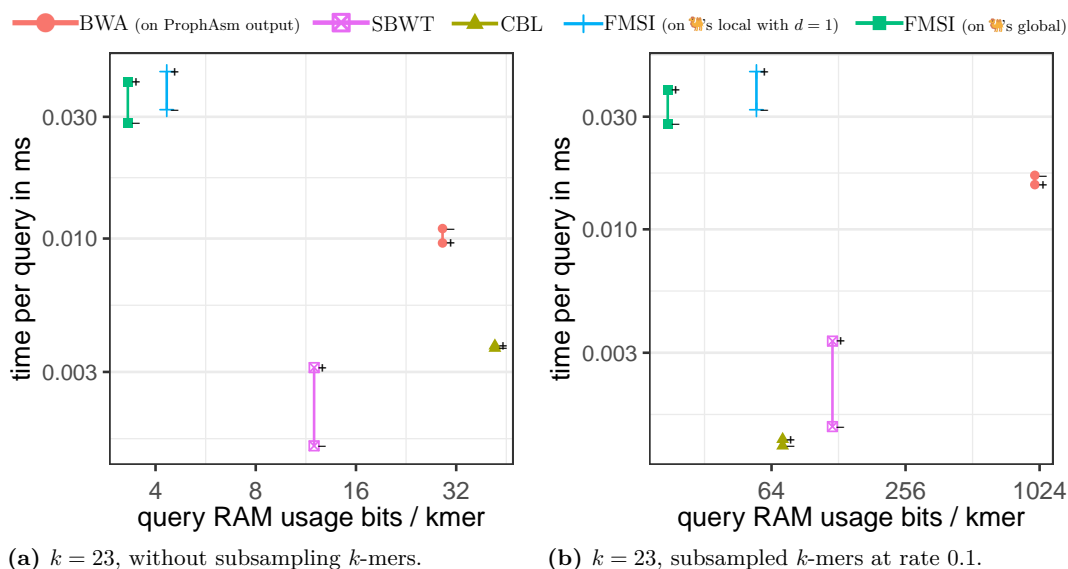
451 Using an indexed f -MS, set operations such as union, (a)symmetric difference, or intersection
452 can be performed directly via their associated abstract operations in Section 4. Indeed, we
453 implement concatenation of masked superstrings via index merging. Prior to concatenating,
454 we only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

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

463 6 Experimental Evaluation

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

469 Index merging is implemented via export, concatenation of the underlying f -MSes, and
470 then reindexing. Compaction is implemented in two variants: first, using k -mer counting and
471 KmerCamel  [61] to construct a superstring; second, using a version of the local algorithm
472 on FMS-index described in Appendix D to work in the bi-directional model at the cost of
473 having $O(k)$ -times higher time complexity. All the demasking functions mentioned in Table 1
474 are supported directly by FMSI, and users can possibly add their custom ones.



■ **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.

475 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 sds1-
 476 lite library [25], available at <https://github.com/simongog/sds1-lite/>, and also uses
 477 KmerCamel 🐫, available at <https://github.com/OndrejSladky/kmercamel>.
 478

479 **Benchmarking methodology.** We evaluated the performance of FMSI (version at commit
 480 a36c2e3) on bacterial and viral pan-genomes and on a nematode genome, both in its efficiency
 481 of construction and querying as well as in performance on set operations, namely computing
 482 intersections, symmetric differences, and unions of k -mer sets. We measured the storage
 483 requirements for each index, both the time and memory requirements for construction and
 484 the time and memory requirements for queries with isolated k -mers. We tested both positive
 485 and negative queries; to generate positive queries, we took a random subset of 10^6 distinct
 486 k -mers from the queried dataset, and to obtain negative queries, we took a random subset of
 487 10^6 distinct k -mers from a part of chromosome 1 of the human genome (genome assembly
 488 GRCh38.p14), excluding those in the queried dataset. All the algorithms were run on a
 489 single thread on a server with AMD EPYC 7302 (3 GHz) processor and 251 GB RAM.

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

- 493 ■ **BWA**¹ [38], a state-of-the-art aligner based on the FM index; for processing queries,
 494 we used the `fastmap` command [36], run with parameter $w = 999999$ on the simplitigs
 495 computed by ProphAsm [11],
- 496 ■ **SBWT**² [1], an index based on the spectral Burrows-Wheeler transform; we used the
 497 default plain-matrix variant as it achieves the best query times in [1] and added all reverse
 498 complements to the index, and

¹ <https://github.com/lh3/bwa>, commit 139f68f.

² <https://github.com/algbio/SBWT>, commit c433b53.

499 ■ CBL³ [45], a very recent method based on smallest cyclic rotations of k -mers.

500 We have run FMSI on the masked superstrings computed by KmerCamel 🐪 [61], specifically
501 the global and local greedy algorithms (local is run with $d = 1$).

502 This experiment was done using an *E. coli* pan-genome (obtained as a union of k -mers of
503 *E. coli* genomes from the 661k collection [5]) and *S. pneumoniae* pan-genome (computed
504 from 616 assemblies from a study of children in Massachusetts, USA [21]) and further verified
505 on a *SARS-CoV-2* pan-genome ($n=14.7$ M, total genome length 430 Gbp). To verify the
506 behavior across diverse datasets, we also provide experimental results for *subsampled* k -mer
507 sets of these three pan-genomes; we note that after subsampling the spectrum-like property
508 (SLP) no longer hold. Specifically, for a given subsampling rate $r \in [0, 1]$, we selected a
509 uniformly random subset of $r \cdot N$ distinct k -mers of the original pan-genome, where N is the
510 total number of k -mers of the pan-genome.

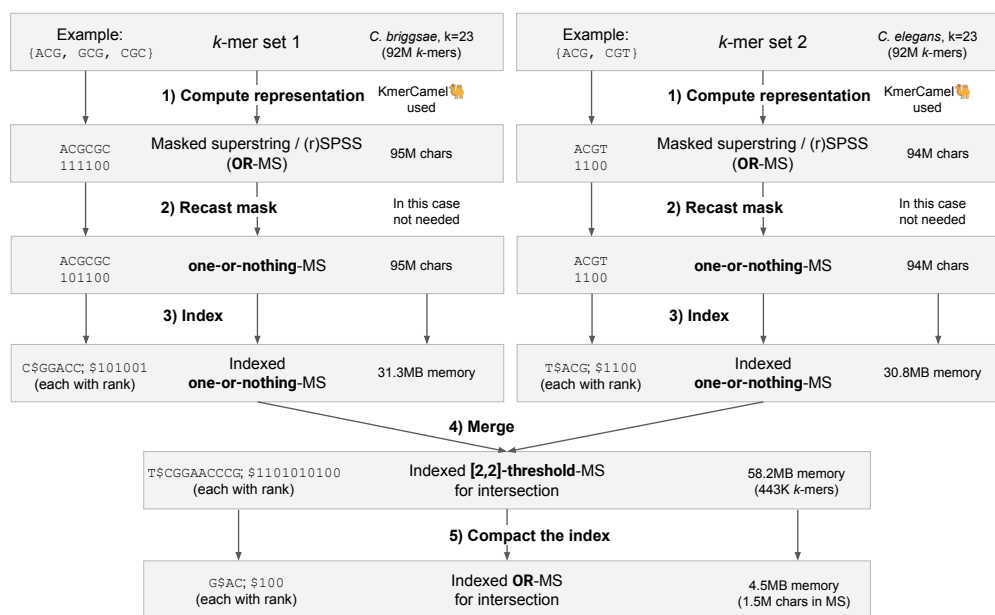
511 The results on the *E. coli* pan-genome for $k = 23$ without subsampling and with
512 subsampling at rate 0.1 are presented in Figure 1; for further results, we refer to the
513 supplementary repository⁴. Across all of the datasets, values of k , and subsampling rates,
514 FMSI run on the masked superstring computed by KmerCamel 🐪's global greedy required
515 3-10 times less memory for processing queries than all of the other methods, attaining
516 around 3-4 bits per k -mer on non-subsampled *E. coli* pan-genome. However, FMSI needed
517 substantially more time for processing queries than the other programs; this is mainly due to
518 the prototype nature of our implementation, and we believe that the query time of FMSI
519 can be substantially optimized. SBWT (in the plain-matrix variant) and CBL are generally
520 the fastest algorithms for processing queries. We note that SBWT required substantial disk
521 space during index construction (up to tens of GBs for the *E. coli* pan-genome). We also
522 remark that while the memory usage per k -mer grows with decreasing subsampling rate, the
523 query time remains roughly the same for all algorithms.

524 **Experimental evaluation of set operations.** We demonstrate the feasibility of using
525 FMSI to perform set operations on k -mer sets. Our proposed pipeline for set operations,
526 as depicted in Figure 2, consists of five steps: First, we compute a textual representation
527 of the k -mer sets interpreted as **or**-masked superstrings. In our experiments, this was
528 done using KmerCamel 🐪's global greedy algorithm. Second, we recast the mask to the
529 desired demasking function, specifically we keep **or** for union and change to **one-or-nothing**
530 for intersection and to **xor** for symmetric difference. In the case of **or**-MS computed by
531 KmerCamel 🐪, mask recasting is actually not needed as the output already minimizes the
532 number of 1s in the mask. Then we index the f -masked superstrings using FMSI (this can be
533 done even before mask recasting). The last two steps are concatenating the two FMS-indexes
534 by index merging and compacting the resulting FMS-index if needed. Note that once indexed,
535 we can ask membership queries on the resulting k -mer sets.

536 For this experiment, we used genomes of *C. elegans* (NC_003279.8, 100M base pairs)
537 and *C. briggsae* (NC_013489.2, 108M base pairs). We evaluated the superstring length of
538 each computed f -masked superstring, and the memory requirements to perform queries
539 on the indexed individual and concatenated f -masked superstrings, both before and after
540 compaction. The results for intersection of the two roundworm genomes for $k = 23$ are
541 depicted in Figure 2. Overall, at every step, the memory required to query the indexed
542 f -masked superstring was around 3 bits per superstring character, which in case of the
543 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 querying the underlying set.

544 the merged FMS-index, albeit for the compacted concatenated result the per- k -mer memory
 545 was higher as it was as low as latent memory required to run FMSI. Note also that the fact
 546 that compaction significantly reduces the masked superstring length highly depends on the
 547 particular use case, namely on the proportion of represented k -mers in the result. For union
 548 and symmetric difference for the same data, the compaction lead to almost negligible length
 549 reduction. For data about symmetric difference and union as well as for other values of k ,
 550 see the supplementary repository⁴.

551 7 Discussion and Conclusions

552 We have proposed f -masked superstrings as an abstract data type for k -mer sets that
 553 allows for seamless execution of set operations. It is primarily based on equipping masked
 554 superstrings from [61] with a demasking function f , and we have studied the effect of using
 555 several natural demasking functions. For symmetric set operations (e.g., union, intersection,
 556 symmetric difference), our approach is efficient even on a larger number of input sets as it
 557 reduces to simple concatenation of masked superstrings and possibly changing the demasking
 558 function. We have also shown how to perform asymmetric operations (e.g., set difference) in
 559 linear time on a constant number of input sets. However, from a practical point of view, the
 560 algorithm for asymmetric operations appears to be less efficient as it requires concatenating
 561 multiple copies of masked superstrings for some input sets. We leave open how to perform
 562 set difference and other asymmetric operations in a more efficient way.

563 Next, we have combined f -masked superstrings with FMS-Index, an adaptation of the

564 FM index and shown how to perform basic operations on the index and process k -mer
565 membership queries efficiently. To demonstrate practicality, we have provided a prototype
566 implementation in a tool called FMSI. While our experimental evaluation already shows
567 significant reduction of memory usage, it still lacks behind state-of-the-art methods in terms of
568 query time. However, we believe that FMSI can be substantially optimized; this is supported
569 by the fact that both FMSI and BWA [38, 36] are based on the FM index and therefore,
570 it should be possible to reduce query times of FMSI on the level of BWA. We also note
571 that we did not optimize other subroutines of FMSI, such as index merging. Furthermore,
572 we leave as an open question how to design efficient algorithms for mask recasting in the
573 suffix-array coordinates and compaction in the bi-directional model without the need to
574 export the f -masked superstring.

575 Our work opens up several research directions for future investigation. First, our main
576 focus was on processing individual k -mer sets that can be combined via set operations and
577 queried for individual k -mers. The support for querying consecutive k -mers of a genomic
578 sequence (e.g., a read obtained from sequencing) can be added by using the bi-directional FM-
579 index [33] (details left to future work). Another significant direction is to extend f -masked
580 superstrings for indexing large collections of k -mer sets (see, e.g., a review in [42]). Finally,
581 our focus was on set operations, while we leave maintaining k -mer sets under insertions and
582 deletions to future work.

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

586 — References —

- 587 1 Jarno N Alanko, Simon J Puglisi, and Jaakko Vuohtoniemi. Small searchable κ -spectra
588 via subset rank queries on the spectral Burrows-Wheeler transform. In *SIAM Conference*
589 *on Applied and Computational Discrete Algorithms (ACDA23)*, pages 225–236. SIAM, 2023.
590 doi:10.1137/1.9781611977714.20.
- 591 2 Jarno N Alanko, Jaakko Vuohtoniemi, Tommi Mäklin, and Simon J Puglisi. Themisto: a
592 scalable colored k -mer index for sensitive pseudoalignment against hundreds of thousands
593 of bacterial genomes. *Bioinformatics*, 39(Supplement_1):i260–i269, 2023. doi:10.1093/
594 bioinformatics/btad233.
- 595 3 Fatemeh Almodaresi, Jamshed Khan, Sergey Madaminov, Michael Ferdman, Rob Johnson,
596 Prashant Pandey, and Rob Patro. An incrementally updatable and scalable system for large-
597 scale sequence search using the Bentley-Saxe transformation. *Bioinformatics*, 38(12):3155–3163,
598 2022. doi:10.1093/bioinformatics/btac142.
- 599 4 Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. Cobs: a compact bit-
600 sliced signature index. In *String Processing and Information Retrieval: 26th International*
601 *Symposium, SPIRE 2019, Segovia, Spain, October 7–9, 2019, Proceedings 26*, pages 285–303.
602 Springer, 2019. doi:10.1007/978-3-030-32686-9_21.
- 603 5 Grace A. Blackwell, Martin Hunt, Kerri M. Malone, Leandro Lima, Gal Horesh, Blaise T. F.
604 Alako, Nicholas R. Thomson, and Zamin Iqbal. Exploring bacterial diversity via a curated
605 and searchable snapshot of archived dna sequences. *PLOS Biology*, 19(11):1–16, 11 2021.
606 doi:10.1371/journal.pbio.3001421.
- 607 6 Alexander Bowe, Taku Onodera, Kunihiro Sadakane, and Tetsuo Shibuya. Succinct de Bruijn
608 graphs. In Benjamin J. Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*
609 *- 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012.*
610 *Proceedings*, volume 7534 of *Lecture Notes in Computer Science*, pages 225–235. Springer,
611 2012. doi:10.1007/978-3-642-33122-0_18.

- 612 7 Phelim Bradley, Henk C Den Bakker, Eduardo PC Rocha, Gil McVean, and Zamin Iqbal.
613 Ultrafast search of all deposited bacterial and viral genomic data. *Nature Biotechnology*,
614 37(2):152–159, 2019.
- 615 8 Phelim Bradley, N Claire Gordon, Timothy M Walker, Laura Dunn, Simon Heys, Bill Huang,
616 Sarah Earle, Louise J Pankhurst, Luke Anson, Mariateresa De Cesare, et al. Rapid antibiotic-
617 resistance predictions from genome sequence data for staphylococcus aureus and mycobacterium
618 tuberculosis. *Nature Communications*, 6(1):10063, 2015. doi:10.1038/ncomms10063.
- 619 9 Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic
620 RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016. doi:10.1038/nbt.3519.
- 621 10 Karel Břinda. Novel computational techniques for mapping and classification of Next-
622 Generation Sequencing data. PhD thesis, Université Paris-Est, 2016. doi:10.5281/zenodo.
623 1045317.
- 624 11 Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scal-
625 able representation of de Bruijn graphs. *Genome Biology*, 22(96), 2021. doi:10.1186/
626 s13059-021-02297-z.
- 627 12 Karel Břinda, Alanna Callendrello, Kevin C Ma, Derek R MacFadden, Themoula Char-
628 alampous, Robyn S Lee, Lauren Cowley, Crista B Wadsworth, Yonatan H Grad, Gregory
629 Kucherov, et al. Rapid inference of antibiotic resistance and susceptibility by genomic neighbour
630 typing. *Nature Microbiology*, 5(3):455–464, 2020. doi:10.1038/s41564-019-0656-6.
- 631 13 Karel Břinda, Leandro Lima, Simone Pignotti, Natalia Quinones-Olvera, Kamil Salikhov,
632 Rayan Chikhi, Gregory Kucherov, Zamin Iqbal, and Michael Baym. Efficient and robust
633 search of microbial genomes via phylogenetic compression. *bioRxiv*, 2023. doi:10.1101/2023.
634 04.15.536996.
- 635 14 Karel Břinda, Kamil Salikhov, Simone Pignotti, and Gregory Kucherov. Prophyle 0.3.1.0.
636 *Zenodo*, 5281, 2017. doi:10.5281/zenodo.5237391.
- 637 15 Rayan Chikhi. K-mer data structures in sequence bioinformatics. HDR thesis, Institut Pasteur
638 Ecole Doctorale “EDITE”, 2021.
- 639 16 Rayan Chikhi, Jan Holub, and Paul Medvedev. Data structures to represent a set of k -long
640 DNA sequences. *ACM Computing Surveys*, 54(1):17:1–17:22, 2022. doi:10.1145/3445967.
- 641 17 Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T. Simpson, and Paul Medvedev. On
642 the representation of de Bruijn graphs. In Roded Sharan, editor, *Research in Computational
643 Molecular Biology*, pages 35–55, Cham, 2014. Springer International Publishing. doi:10.1007/
644 978-3-319-05269-4_4.
- 645 18 Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from
646 sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016. doi:
647 10.1093/bioinformatics/btw279.
- 648 19 Thomas C. Conway and Andrew J. Bromage. Succinct data structures for assembling large
649 genomes. *Bioinformatics*, 27(4):479–486, 2011. doi:10.1093/bioinformatics/btq697.
- 650 20 Victoria G. Crawford, Alan Kuhnle, Christina Boucher, Rayan Chikhi, and Travis Gagie.
651 Practical dynamic de Bruijn graphs. *Bioinformatics*, 34(24):4189–4195, 2018. doi:10.1093/
652 bioinformatics/bty500.
- 653 21 Nicholas J. Croucher, Jonathan A. Finkelstein, Stephen I. Pelton, Julian Parkhill, Stephen D.
654 Bentley, Marc Lipsitch, and William P. Hanage. Population genomic datasets describing the
655 post-vaccine evolutionary epidemiology of streptococcus pneumoniae. *Scientific Data*, 2, 2015.
656 doi:10.1038/sdata.2015.58.
- 657 22 Jason Fan, Jamshed Khan, Giulio Ermanno Pibiri, and Rob Patro. Spectrum preserving
658 tilings enable sparse and modular reference indexing. In Haixu Tang, editor, *Research in
659 Computational Molecular Biology - 27th Annual International Conference, RECOMB 2023,
660 Istanbul, Turkey, April 16-19, 2023, Proceedings*, volume 13976 of *Lecture Notes in Computer
661 Science*, pages 21–40. Springer, 2023. doi:10.1007/978-3-031-29119-7\2.

- 662 23 Jason Fan, Jamshed Khan, Noor Pratap Singh, Giulio Ermanno Pibiri, and Rob Patro. Fulgor:
663 a fast and compact k-mer index for large-scale matching and color queries. *Algorithms for*
664 *Molecular Biology*, 19(1):3, 2024. doi:10.1186/S13015-024-00251-9.
- 665 24 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581,
666 2005. doi:10.1145/1082036.1082039.
- 667 25 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug
668 and play with succinct data structures. In *13th International Symposium on Experimental*
669 *Algorithms, (SEA 2014)*, pages 326–337, 2014.
- 670 26 Gaurav Gupta, Minghao Yan, Benjamin Coleman, Bryce Kille, R. A. Leo Elworth, Tharun
671 Medini, Todd Treangen, and Anshumali Shrivastava. Fast processing and querying of 170TB of
672 genomics data via a Repeated And Merged BloOm filter (RAMBO). In *Proceedings of the 2021*
673 *International Conference on Management of Data, SIGMOD '21*, page 2226–2234, New York,
674 NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3448016.3457333.
- 675 27 Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of
676 colored and compacted de Bruijn graphs. *Genome Biology*, 21(1):1–20, 2020. doi:10.1186/
677 s13059-020-02135-8.
- 678 28 James Holt and Leonard McMillan. Merging of multi-string bwts with applications. *Bioinfor-*
679 *matics*, 30(24):3524–3531, August 2014. URL: [http://dx.doi.org/10.1093/bioinformatics/](http://dx.doi.org/10.1093/bioinformatics/btu584)
680 [btu584](http://dx.doi.org/10.1093/bioinformatics/btu584), doi:10.1093/bioinformatics/btu584.
- 681 29 Lauris Kaplinski, Maarja Lepamets, and Maido Remm. GenomeTester4: a toolkit for per-
682 forming basic set operations - union, intersection and complement on k-mer lists. *GigaScience*,
683 4(1):s13742–015–0097–y, 12 2015. doi:10.1186/s13742-015-0097-y.
- 684 30 Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Christopher Barber, Marc Zimmermann,
685 Gunnar Rätsch, and André Kahles. Metagraph: Indexing and analysing nucleotide archives at
686 petabase-scale. *BioRxiv*, pages 2020–10, 2020. doi:10.1101/2020.10.01.322164.
- 687 31 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. *Linear-Time*
688 *Longest-Common-Prefix Computation in Suffix Arrays and Its Applications*, pages 181–192.
689 Springer Berlin Heidelberg, 2001. doi:10.1007/3-540-48194-x_17.
- 690 32 Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating
691 k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 05 2017. doi:10.1093/bioinformatics/
692 btx304.
- 693 33 T. W. Lam, Ruiqiang Li, Alan Tam, Simon Wong, Edward Wu, and S. M. Yiu. High
694 throughput short read alignment via bi-directional bwt. In *2009 IEEE International Conference*
695 *on Bioinformatics and Biomedicine*. IEEE, 2009. doi:10.1109/bibm.2009.42.
- 696 34 John A Lees, Marco Galardini, Stephen D Bentley, Jeffrey N Weiser, and Jukka Corander.
697 pyseer: a comprehensive tool for microbial pangenome-wide association studies. *Bioinformatics*,
698 34(24):4310–4312, 2018. doi:10.1093/bioinformatics/bty539.
- 699 35 Téo Lemane, Paul Medvedev, Rayan Chikhi, and Pierre Peterlongo. kmtricks: efficient and
700 flexible construction of Bloom filters for large sequencing data collections. *Bioinformatics*
701 *Advances*, 2(1):vbac029, 04 2022. doi:10.1093/bioadv/vbac029.
- 702 36 Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly.
703 *Bioinformatics*, 28(14):1838–1844, 2012. doi:10.1093/bioinformatics/bts280.
- 704 37 Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM.
705 *arXiv preprint arXiv:1303.3997*, 2013. doi:10.48550/arXiv.1303.3997.
- 706 38 Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler
707 transform. *Bioinformatics*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.
- 708 39 Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows-Wheeler
709 transform. *Bioinformatics*, 26(5):589–595, 2010. doi:10.1093/bioinformatics/btp698.
- 710 40 Antoine Limasset, Guillaume Rizk, Rayan Chikhi, and Pierre Peterlongo. Fast and scalable
711 minimal perfect hashing for massive key sets. In Costas S. Iliopoulos, Solon P. Pissis, Simon J.
712 Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms*,

- 713 *SEA 2017, June 21-23, 2017, London, UK*, volume 75 of *LIPICs*, pages 25:1–25:16. Schloss
714 Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/lipics.sea.2017.25.
- 715 **41** Po-Ru Loh, Michael Baym, and Bonnie Berger. Compressive genomics. *Nature Biotechnology*,
716 30(7):627–630, July 2012. URL: <http://dx.doi.org/10.1038/nbt.2241>, doi:10.1038/nbt.
717 2241.
- 718 **42** Camille Marchet, Christina Boucher, Simon J Puglisi, Paul Medvedev, Mikaël Salson, and
719 Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing
720 data sets. *Genome Research*, 31(1):1–12, 2021. doi:10.1101/gr.260604.119.
- 721 **43** Camille Marchet, Zamin Iqbal, Daniel Gautheret, Mikaël Salson, and Rayan Chikhi. REIN-
722 DEER: efficient indexing of k-mer presence and abundance in sequencing datasets. *Bioinfor-*
723 *matics*, 36(Supplement-1):i177–i185, 2020. doi:10.1093/bioinformatics/btaa487.
- 724 **44** Camille Marchet, Maël Kerbiriou, and Antoine Limasset. Blight: efficient exact associative
725 structure for k-mers. *Bioinformatics*, 37(18):2858–2865, 2021. doi:10.1093/bioinformatics/
726 btab217.
- 727 **45** Igor Martayan, Bastien Cazaux, Antoine Limasset, and Camille Marchet. Conway-Bromage-
728 Lyndon (CBL): an exact, dynamic representation of k-mer sets. *bioRxiv*, 2024. doi:10.1101/
729 2024.01.29.577700.
- 730 **46** Martin D. Muggli, Bahar Alipanahi, and Christina Boucher. Building large updatable
731 colored de Bruijn graphs via merging. *Bioinformatics*, 35(14):i51–i60, 2019. doi:10.1093/
732 bioinformatics/btz350.
- 733 **47** Martin D. Muggli, Alexander Bowe, Noelle R. Noyes, Paul S. Morley, Keith E. Belk, Robert
734 Raymond, Travis Gagie, Simon J. Puglisi, and Christina Boucher. Succinct colored de Bruijn
735 graphs. *Bioinformatics*, 33(20):3181–3187, 2017. doi:10.1093/bioinformatics/btx067.
- 736 **48** Gonzalo Navarro. *Texts*, page 395–449. Cambridge University Press, 2016.
- 737 **49** Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. Squeakr: an exact
738 and approximate k-mer counting system. *Bioinformatics*, 34(4):568–575, 2018. doi:
739 10.1093/bioinformatics/btx636.
- 740 **50** Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides
741 fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419,
742 2017. doi:10.1038/nmeth.4197.
- 743 **51** Giulio Ermanno Pibiri. Sparse and skew hashing of K-mers. *Bioinformatics*, 38(Supple-
744 ment_1):i185–i194, 2022. doi:10.1093/bioinformatics/btac245.
- 745 **52** Amatur Rahman. Compression algorithms for de Bruijn graphs and uncovering hidden
746 assembly artifacts. PhD thesis, The Pennsylvania State University, 2023.
- 747 **53** Amatur Rahman and Paul Medvedev. Representation of k-mer sets using spectrum-preserving
748 string sets. *Journal of Computational Biology*, 28(4):381–394, 2021. PMID: 33290137. doi:
749 10.1089/cmb.2020.0431.
- 750 **54** Kamil Salikhov. Efficient algorithms and data structures for indexing dna sequence data. PhD
751 thesis, Université Paris-Est, 2017.
- 752 **55** Kamil Salikhov, Karel Břinda, Simone Pignotti, and Gregory Kucherov. ProPhex. <https://github.com/prophyle/prophex>. doi:10.5281/zenodo.1247432.
- 753 **56** Sebastian Schmidt. Unitigs are not enough: the advantages of superunitig-based algorithms
754 in bioinformatics. PhD thesis, University of Helsinki, 2023.
- 755 **57** Sebastian Schmidt and Jarno N. Alanko. Eulertigs: minimum plain text representation of
756 k-mer sets without repetitions in linear time. *Algorithms for Molecular Biology*, 18(1):5, 2023.
757 doi:10.1186/s13015-023-00227-1.
- 758 **58** Sebastian Schmidt, Shahbaz Khan, Jarno N. Alanko, Giulio E. Pibiri, and Alexandru I.
759 Tomescu. Matchtigs: minimum plain text representation of k-mer sets. *Genome Biology*,
760 24(1):136, 2023. doi:10.1186/s13059-023-02968-z.
- 761 **59** Liam P Shaw, Eduardo P C Rocha, and R Craig MacLean. Restriction-modification systems
762 have shaped the evolution and distribution of plasmids across bacteria. *Nucleic Acids Research*,
763 51(13):6806–6818, 2023. doi:10.1093/nar/gkad452.
- 764

- 765 **60** Yoshihiro Shibuya, Djamel Belazzougui, and Gregory Kucherov. Efficient Reconciliation of
766 Genomic Datasets of High Similarity. In Christina Boucher and Sven Rahmann, editors, *22nd*
767 *International Workshop on Algorithms in Bioinformatics (WABI 2022)*, volume 242 of *Leibniz*
768 *International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:14, Dagstuhl, Germany, 2022.
769 Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.WABI.2022.14.
- 770 **61** Ondřej Sladký, Pavel Veselý, and Karel Břinda. Masked superstrings as a unified framework
771 for textual k-mer set representations. *bioRxiv*, 2023. doi:10.1101/2023.02.01.526717.
- 772 **62** Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J
773 Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data:
774 Astronomical or genetical? *PLoS Biology*, 13(7):e1002195, 2015. doi:10.1371/journal.
775 pbio.1002195.
- 776 **63** Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification
777 using exact alignments. *Genome Biology*, 15(3):1–12, 2014. doi:10.1186/gb-2014-15-3-r46.

778 **A Proof of Uniqueness of Union Function**

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

785 ► **Theorem 10.** *or is the only comprehensive demasking function f such that for any*
 786 *two k -mer sets K and K' and any of their valid f -masked superstrings (f, S, M) and*
 787 *(f, S', M') , respectively, their concatenation $(f, S+S', M+M')$ is a valid f -masked superstring*
 788 *representing the set $K \cup K'$.*

789 **Proof of Theorem 10.** For a contradiction assume there is a comprehensive demasking
 790 function f different than **or** that satisfies the above. Consider the smallest n such that f
 791 behaves differently than **or** for a length- n input, meaning that there is $x \in \{0, 1\}^n$ not equal
 792 to the all-zeros vector such that $f(x) \neq 1$. As f is comprehensive, it cannot happen that
 793 $f(x) = 1$ for all $x \in \{0, 1\}^n$ by Definition 4, and moreover, $f((1)) = 1$, implying $n > 1$. Fix
 794 a k -mer, for simplicity A^k (although similar approach works for all k -mers). We take the
 795 first f -masked superstring to be the k -mer with mask being $M_0 = x_0$ and $M_i = 0$ for the
 796 remaining $k - 1$ positions $i > 0$. The second f -masked superstring is $CA \dots A$ where A appears
 797 $n + k - 2$ times with the mask being: $M_0 = 0$, $M_i = x_i$ for $i = 1, \dots, k$, and $M_i = 0$ for $i > k$.
 798 At least one of the represented sets contains the k -mer as $x \neq 0$ and n is the but the resulting
 799 f -masked superstring is either invalid or does not contain the k -mer in the represented set as
 800 $f(x) \neq 1$, a contradiction. ◀

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

804 **Proof of Theorem 11.** It is sufficient to find a construction of matchtigs such that we can
 805 construct an arbitrary sequence of ones and zeros at the occurrences of a given k -mer and
 806 the rest follows similarly as in Theorem 10.

807 We do this with k -mer CG and matchtigs Cg and Gc . Consider the counterexample sequence
 808 of occurring ones and zeros from Theorem 10. For every one in the sequence, we add the
 809 matchtig Cg and for each m consecutive zeros, we add $m + 1$ times the matchtig Gc , since
 810 at the boundary of two Gc matchtigs an OFF occurrence of k -mer CG appears. At any other
 811 boundary, the k -mer CG does not appear, therefore the construction is correct. The rest of
 812 the proof follows a similar argument as in Theorem 10. ◀

813 Note, however, that the same does not hold if we want to represent simplitigs/SPSS
 814 solely. As an individual k -mer cannot appear more than once with an ON occurrence, any
 815 comprehensive function generalizes SPSS representations if it satisfies that if there is one ON
 816 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. ◀

859 **C** Alternative Demasking Functions

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

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

863 Perhaps the simplest approach to representing a set of *k*-mers is to mark all occurrences of
864 represented *k*-mers with one, all ghost *k*-mers with zero, and treat all other masks as invalid.
865 This corresponds to a function that returns 1 if it receives a list of ones, 0 if a list of zeros
866 (or an empty list), and `invalid` otherwise.

867 This representation has its clear benefits. Most importantly, one can determine the
868 presence or absence of a *k*-mer by looking at the mask at any occurrence of the *k*-mer. For
869 example, this makes indexing of **all-or-nothing**-masked superstrings easier than indexing
870 general *f*-masked superstrings, as we do not need to query the rank to determine the number
871 of ON occurrences. Instead, we can simply determine the presence or absence of a *k*-mer
872 based on any of its occurrences, as discussed in Section 5.3.

873 We could potentially achieve higher compressibility of the mask by realizing that we
874 can infer the presence or absence of a *k*-mer from its first occurrence, which comes from
875 the fact that a mask for a given set is unique. Thus, we can omit all symbols in the mask
876 corresponding to any further occurrences of the *k*-mer, making the mask shorter and easier
877 to store, while it can be easily reconstructed afterwards.

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

880 **C.2** The and-masked superstrings

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

886 The potential advantage of **and**-masked superstrings over **or**-masked superstrings is that
887 we can mark ghost *k*-mers with ones at some occurrences and therefore obtain masks with
888 more ones in them, which could be beneficial, for instance, for additional improvements in
889 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.