



HAL
open science

Efficient reconciliation of genomic datasets of high similarity

Yoshihiro Shibuya, Djamel Belazzougui, Gregory Kucherov

► **To cite this version:**

Yoshihiro Shibuya, Djamel Belazzougui, Gregory Kucherov. Efficient reconciliation of genomic datasets of high similarity. 22nd International Workshop on Algorithms in Bioinformatics (WABI 2022), Sep 2022, Potsdam, Germany. 10.4230/LIPIcs.WABI.2022.14 . hal-03867538

HAL Id: hal-03867538

<https://cnrs.hal.science/hal-03867538>

Submitted on 23 Nov 2022

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.

Efficient reconciliation of genomic datasets of high similarity

Yoshihiro Shibuya  

LIGM, Université Gustave Eiffel, Marne-la-Vallée, France

Djamal Belazzougui 

CAPA, DTISI, Centre de Recherche sur l'Information Scientifique et Technique, Algiers, Algeria

Gregory Kucherov  

LIGM, CNRS, Université Gustave Eiffel, Marne-la-Vallée, France

Abstract

We apply Invertible Bloom Lookup Tables (IBLTs) to the comparison of k -mer sets originated from large DNA sequence datasets. We show that for similar datasets, IBLTs provide a more space-efficient and, at the same time, more accurate method for estimating Jaccard similarity of underlying k -mer sets, compared to MinHash which is a go-to sketching technique for efficient pairwise similarity estimation. This is achieved by combining IBLTs with k -mer sampling based on syncmers, which constitute a context-independent alternative to minimizers and provide an unbiased estimator of Jaccard similarity. A key property of our method is that involved data structures require space proportional to the difference of k -mer sets and are independent of the size of sets themselves. As another application, we show how our ideas can be applied in order to efficiently compute (an approximation of) k -mers that differ between two datasets, still using space only proportional to their number. We experimentally illustrate our results on both simulated and real data (*SARS-CoV-2* and *Streptococcus Pneumoniae* genomes).

2012 ACM Subject Classification Applied computing

Keywords and phrases k -mers, sketching, Invertible Bloom Lookup Tables, IBLT, MinHash, syncmers, minimizers

Digital Object Identifier 10.4230/LIPIcs.WABI.2022.14

Related Version <https://doi.org/10.1101/2022.06.07.495186>

Supplementary Material <https://github.com/yhshhb/km-peeler.git>

1 Introduction

Alignment-free methods became a prevalent paradigm in computational analysis of modern genomic datasets. However, despite being faster than their alignment-based counterparts, algorithms based on k -mer sets are starting to struggle when applied to the large datasets produced nowadays [20, 13, 16]. To deal with this issue, a considerable effort has been put to developing optimized data structures, with succinct solutions [25, 23, 16] and approximate membership data structures [29, 13, 2, 14, 3] being two examples.

In recent years, sketching techniques have been gaining increasing attention thanks to their capacity of drastically decreasing space usage. MinHash is probably the most well-known representative of this family of algorithms. Application of MinHash to comparison of DNA sequence datasets was pioneered in Mash software [24] and subsequently used in several other tools. With this approach, input datasets are transformed into smaller “sketches” on which subsequent comparisons are performed. In short, sequences are first fragmented into their constituent k -mers which are then hashed, with each sketch storing only s minimum values, with s defined by the user. The fraction of shared hashes between two sketches is an



© Yoshihiro Shibuya and Djamal Belazzougui and Gregory Kucherov;
licensed under Creative Commons License CC-BY 4.0

22nd International Workshop on Algorithms in Bioinformatics (WABI 2022).

Editors: Christina Boucher and Sven Rahmann; Article No. 14; pp. 14:1–14:15

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

44 unbiased estimator of the Jaccard similarity index [4]. A MinHash sketch can thus be viewed
45 as a sample of the set of k -mers of the sequence it represents. Given that s is much smaller
46 than the genome length, working with the sampled hashes leads to fast pairwise comparisons
47 using small memory. However, when two sequences are close and share most of their k -mers,
48 MinHash sketches of small size are not able to reliably estimate their degree of similarity
49 since differences are likely to be missed during sampling.

50 In this work, we propose an alternative approach to evaluate the difference in k -mer
51 composition of two related datasets. Our method relies on Invertible Bloom Lookup Table
52 (IBLT) data structure [12, 10] which is an extension of Bloom filters, supporting deletions of
53 items and, most importantly, enumeration (with high probability) of stored items. One of
54 the applications of IBLT is reconciliation of two sets of items: in a scenario considered in
55 [12], a set A is stored in an IBLT which is then transmitted to the holder of another set B .
56 By screening B against the IBLT of A it is possible to recover the items $A \setminus B$ and $B \setminus A$,
57 with high probability. This is done through the so-called *peeling* procedure [7].

58 In this paper we make one step further: inspired by ideas of [26], we recover both $A \setminus B$
59 and $B \setminus A$ from IBLTs of A and B , rather than from an IBLT of one of them and the whole
60 other set. Furthermore, a crucial property is that the *size of these IBLTs is bounded in*
61 *terms of the symmetric difference size $(A \setminus B) \cup (B \setminus A)$* rather than the size of the original
62 sets. This provides a key to the efficiency of our solution when input sets are similar: even
63 if input sets are very big, their difference can be recovered using a data structure (sketch)
64 whose size is proportional to the size of the difference of those sets rather than of the sets
65 themselves. Estimating the symmetric difference allows us to estimate the Jaccard similarity,
66 using information about the sizes of input sets. Thus, whereas close datasets require larger
67 MinHash sketches to be properly compared, our method, on the contrary, requires smaller
68 memory.

69 Another ingredient of our solution is k -mer sampling. Intuitively, since two adjacent k -
70 mers share $k-1$ bases, the information stored in the set of all k -mers appears highly redundant.
71 One popular method of sampling k -mers from genomic sequences is based on minimizers
72 [28]. Under this technique, consecutive sampled k -mers are within a bounded distance from
73 each other and therefore no large portion of the sequence can remain unsampled. Another
74 favorable property is that similar regions are likely to yield similar samples of minimizers.
75 However, it has recently been shown that estimating Jaccard similarity based on minimizer
76 sampling leads to a bias [1]. Here we propose to replace minimizers by syncmers [8]. Syncmers
77 provide another way of k -mer sampling which has certain advantages over minimizers. As
78 opposed to minimizers, syncmers are not context-dependent: for a k -mer to be a syncmer
79 depends on the k -mer alone regardless the context where it occurs, and, under standard
80 randomness assumptions on involved hash functions, all k -mers have equal chance to be a
81 syncmer. As a consequence, syncmer sampling leads to an unbiased estimate of Jaccard
82 similarity, as the fraction of syncmers among shared k -mers (intersection) is expected to be
83 the same as that among all k -mers (union). We experimentally validate that this is indeed
84 the case.

85 By combining syncmer sampling with IBLTs, we obtain a space-efficient method for
86 accurately estimating Jaccard similarity for similar datasets. For datasets of high similarity,
87 the proposed method is superior to the popular MinHash algorithm [24], both in terms of
88 memory and precision. We also propose an application of this technique to retrieve k -mers
89 that differ between two given datasets. Our method computes a superset of those k -mers with
90 a limited number of spurious k -mers. In particular, under the assumption that each k -mer
91 occurs once, our method computes the exact set differences between involved k -mer sets. We

92 validate our algorithms on both simulated data and on real datasets made of *SARS-CoV-2*
 93 and *Staphylococcus Pneumoniae* genomes.

94 **2** Technical preliminaries

95 We consider DNA alphabet $\Sigma = \{A, C, G, T\}$ even though our algorithms can be easily
 96 generalized. Given a string $S \in \Sigma^*$, we use the notation $S[i, k]$ to indicate the substring of
 97 length k starting at position i called a k -mer. The k -mer set K_S of S is the set of k -mers
 98 $S[i, k]$ for $i \in [0, |S| - k + 1]$.

99 **2.1** Minimizers

100 Independently introduced in [28] and [30], minimizers are defined by a triplet of parameters
 101 (k, w, h) , where k is the k -mer length, w a window size, and h a function defining an order on
 102 k -mers. h is usually chosen to be an appropriately defined hash function, the lexicographical
 103 order is rarely used in practice due to its poor statistical properties.

104 Each window $S[i, w + k - 1]$ defines a minimizer which is the minimal k -mer among w
 105 k -mers occurring in $S[i, w + k - 1]$ w.r.t. the order given by h . Two neighboring minimizers
 106 are thus separated by at most w positions making it impossible to have large stretches of the
 107 original sequence not covered by any minimizers.

108 Since two neighboring windows at positions i and $i + 1$ are likely to share their minimizer,
 109 minimizers provide a way to sample k -mers from a sequence with bounded distance between
 110 consecutive sampled k -mers. An advantage of this sampling strategy is that similar sequences
 111 will likely have similar lists of minimizers, which makes it useful for mapping algorithms
 112 [19, 15]. Under reasonable assumptions, the density of minimizers, i.e. fraction of sampled
 113 k -mers, is $\frac{2}{w+1}$ [28, 8]. If minimizer positions in the original sequence are not important,
 114 they can be discarded and the resulting k -mer multiset can be reduced to a simple k -mer set.

115 **2.2** Syncmers

116 Minimizers are susceptible to mutations of any base of their window [8]. That is, a k -mer
 117 may cease to be a minimizer if a modified base occurs not only inside this k -mer, but also
 118 in its close neighborhood. Sampling with a higher density alleviates this problem but it
 119 reduces the advantages of the methods because more minimizers are selected. Methods to
 120 generate minimizer indices with the best possible density exist [6, 9] but they are usually
 121 offline algorithms, limiting their potential applications outside alignment.

122 Syncmers are a family of alternative methods to minimizers that does not suffer from this
 123 issue [8]. Similarly to minimizers, syncmers are defined using a triplet of parameters (k, z, h)
 124 where $z < k$ is used to decompose each k -mer into its constituent z -mers and h defines an
 125 order over them. A k -mer q is a *syncmer* (called *closed syncmers* in [8]) iff its minimal z -mer
 126 occurs as a prefix (position $i = 0$) or as a suffix (position $i = k - z + 1$) of q . Thus, a syncmer
 127 is defined by its sequence alone, regardless the context in which it occurs. For this reason,
 128 syncmer sampling has been shown to be more resistant to mutations and then to improve
 129 the sensitivity of alignment algorithms [8].

130 Similar to minimizers, consecutive syncmers occur at a bounded distance. More precisely,
 131 consecutive syncmers must overlap by at least z characters and therefore “pave” the sequence
 132 without gaps. The fraction of syncmers among all k -mers is estimated to be $\frac{2}{k-z+1}$ [8].

133 2.3 Invertible Bloom Lookup Tables

134 Invertible Bloom Lookup Tables (IBLT) [10, 12] are a generalization of Bloom filters for
 135 storing a set of elements (keys), drawn from a large universe, possibly associated with
 136 attribute values. In contrast to Bloom filters, in addition to insertions, IBLTs also support
 137 deletion of keys as well as listing. The latter operation succeeds with high probability (w.h.p.)
 138 depending on the number of stored keys relative to the size of the data structure. An
 139 important property is that this probability depends only on the number of keys stored at the
 140 moment of listing, and not across the entire lifespan of the data structure. Thus, at a given
 141 time, an IBLT can store a number of keys greatly exceeding the threshold for which it was
 142 built, returning to be fully functional whenever a sufficient number of deletions has taken
 143 place. Note also that IBLTs, in their basic version, don't support multiple insertions of the
 144 same key.

145 An IBLT is an array T of m buckets together with r hash functions h_1, \dots, h_r mapping
 146 a key universe U (in our case, k -mers or strings) to $[0..m - 1]$ and an additional global hash
 147 function h_e on U . Each bucket $T[i], i \in [0..m - 1]$, contains three fields: a counter $T[i].C$,
 148 a key field $T[i].P$ and a hash field $T[i].H$, where C counts the number of keys hashed to
 149 bucket i , P stores the XOR-sum of the keys (in binary representation) hashed to bucket i ,
 150 and H contains the XOR-sum of hashes produced by h_e on keys.

151 Adding a key p to the IBLT is done as follows. For each $j \in \{1, \dots, r\}$, we perform
 152 $T[h_j(p)].C = T[h_j(p)].C + 1$, $T[h_j(p)].P = T[h_j(p)].P \oplus p$, and $T[h_j(p)].H = T[h_j(p)].H \oplus$
 153 $h_e(p)$, where \oplus stands for XOR. Given that XOR is the inverse operation of itself, deletion
 154 of p is done similarly except that $T[h_j(p)].C = T[h_j(p)].C - 1$.

155 Listing the keys held in an IBLT is done through the process of peeling working recursively
 156 as follows. If for some i we have $T[i].C = 1$, payload field $T[i].P$ is supposed to contain
 157 a single key p . Field H is not strictly necessary, it acts as a “checksum” to verify that p
 158 is indeed a valid key by checking if $h_e(T[h_j(p)].P) = T[h_j(p)].H$. This check is used to
 159 avoid the case when $T[i].C = 1$ whereas $T[i].P$ is not a valid key, which can result from
 160 extraneous deletions of keys not present in the data structure. In Section 3.2 we will elaborate
 161 on the role of this field in our framework. If the check holds, key p can be reported and
 162 deleted (peeled) from the IBLT. Updating hash sums and counters is done in a similar way:
 163 $T[h_j(p)].H = T[h_j(p)].H \oplus h_e(p)$ and $T[h_j(p)].C = T[h_j(p)].C - 1$. The procedure continues
 164 until all counters $T[i].C$ are equal to zero.

165 At each moment, an IBLT is associated to a r -hypergraph where nodes are buckets and
 166 edges correspond to stored keys with each edge including the buckets a key is hashed to. Listing
 167 the keys contained in an IBLT then relies on the peelability property of random hypergraphs
 168 [7, 22]. Assume our hash functions are fully random. Then it is known that for $r \geq 3$, a
 169 random r -hypergraph with m nodes and n edges is peelable w.h.p. iff $m \geq c_r n$ where c_r is a
 170 constant peelability threshold. The first values of c_r are $c_3 \approx 1.222, c_4 \approx 1.295, c_5 \approx 1.425, \dots$
 171 [12]. Thus, allocating

$$172 \quad m = n(c_r + \varepsilon), \tag{1}$$

173 buckets, for $\varepsilon > 0$, for storing n keys guarantees successful peeling with high probability.

174 2.4 MinHash sketching

175 MinHash sketching was introduced in [4] as a method to estimate Jaccard similarity between
 176 two sets, applied to document comparison. In bioinformatics, MinHash was first applied in
 177 Mash software [24] and then successfully used in a number of other tools. Assume we are

178 given a universe U and an order on U defined via a hash function h . For a set $A \subset U$, the
 179 bottom- s MinHash sketch of A , denoted $\mathbb{S}(A)$, is the set of s minimal elements of A (or their
 180 hashes), where s is a user-defined parameter. The Jaccard similarity index between two sets
 181 A and B , $J(A, B) = |A \cap B|/|A \cup B|$, can then be estimated from the sketches of A and B ,
 182 namely

$$183 \quad |\mathbb{S}(A \cap B) \cap \mathbb{S}(A) \cap \mathbb{S}(B)|/|\mathbb{S}(A \cup B)| \quad (2)$$

184 is an unbiased estimator of $J(A, B)$.

185 The Jaccard similarity between the k -mer sets of two dataset constitutes a biologically
 186 relevant measure of their similarity. In particular, if involved datasets are genomic sequences,
 187 this measure allows one to estimate the mutation rate between the sequences [11, 24].

188 **3 Methods**

189 **3.1 Set reconciliation from two IBLTs**

190 Invertible Bloom Lookup Tables can be used to achieve set reconciliation between two sets
 191 A and B , that is to recover sets $A \setminus B$ and $B \setminus A$. Under a scenario described in [12], the
 192 holder of A stores it in an IBLT T_A which is then transmitted to the holder of B . Elements
 193 of B are then deleted from T_A . In the resulting IBLT, P -fields with $T_A[i].C = 1$ correspond
 194 to elements of $A \setminus B$ and those with $T_A[i].C = -1$ to $B \setminus A$. The peeling process is applied
 195 to either of such fields. Whenever $T_A[i].C = 1$, we delete $p = T_A[i].P$ from T_A on condition
 196 that $h_e(p) = T_A[i].H$. Similarly, whenever $T_A[i].C = -1$, we add (XOR) $p = T_A[i].P$ to T_A
 197 on condition that $h_e(p) = T_A[i].H$. The process lists all elements of both $A \setminus B$ and $B \setminus A$
 198 w.h.p.

199 Inspired by work [26], we modify the above scheme in order to recover the symmetric
 200 difference between A and B from their respective IBLTs T_A and T_B , rather than from the
 201 IBLT of one set and the whole other set. To do this, we define T_{A-B} and T_B to be of the same size
 202 and to use the same hash functions. We then compute the difference of T_A and T_B , denoted
 203 T_{A-B} and defined through $T_{A-B}[i].C = T_A[i].C - T_B[i].C$, $T_{A-B}[i].P = T_A[i].P \oplus T_B[i].P$,
 204 and $T_{A-B}[i].H = T_A[i].H \oplus T_B[i].H$. Information about elements of $A \cap B$ is “cancelled out”
 205 in T_{A-B} , that is, T_{A-B} holds elements of $(A \setminus B) \cup (B \setminus A)$. Peeling then proceeds as usual,
 206 listing both $A \setminus B$ and $B \setminus A$ with the distinction made possible by looking at the sign of C .

207 A remarkable property of this scheme is that it allows one to recover set differences using
 208 a space proportional to the size of those differences regardless the size of the involved sets.
 209 Indeed, for the peeling process to succeed w.h.p., it is sufficient that the size of T_{A-B} be $O(n)$
 210 where $n = |(A \setminus B) \cup (B \setminus A)|$ (see (1)). This is particularly suitable for the bioinformatics
 211 framework where we are often dealing with highly similar datasets, such as genomes of
 212 different individuals or closely related species.

213 **3.2 Making buckets lighter**

214 In the above scheme of IBLT difference, the H field becomes important as the case
 215 $T_{A-B}[i].C = 1$ (or $T_{A-B}[i].C = -1$) can occur due to a spurious “cancelling out” of
 216 distinct keys. However, to save space, we propose to get rid of the H field and replace the
 217 “checksum” verification by another test: if $T_{A-B}[i].C = 1$ (resp. $T_{A-B}[i].C = -1$), we check
 218 whether $p = T_{A-B}[i].P$ is a valid key by checking if $h_j(p) = i$ for one of $j \in [1..r]$. This
 219 allows us to save space at the price of additional verification time. This technique works

220 particularly well for large IBLTs but it becomes less effective for small ones, as the “false
221 positive” probability is proportional to the size of the table.

222 3.3 Combining sampling and IBLTs for Jaccard similarity estimation

223 We now turn to our main goal: estimating Jaccard similarity of two k -mer sets using IBLTs.
224 The common approach uses MinHash sketching as described in [24] (see Section 2.4). However,
225 MinHash requires larger sketches to measure similarity of close datasets. One possible idea
226 could be to store MinHash sketches in IBLTs in hope to use them for estimating Jaccard
227 similarity through the IBLT-difference scheme from the previous section. This, however, runs
228 into an obstacle due to the fact that applying (2) requires knowledge of k -mers belonging to
229 the sketch intersection, and not only to sketch differences.

230 Rather than working with the entire sets of k -mers, we resort to sampling. It is known
231 that sampling minimizers incurs a bias in estimating Jaccard similarity [1]. Instead, we
232 propose to use syncmers, which don’t suffer from being context-dependent thus resulting in
233 an unbiased estimator of Jaccard similarity.

234 To justify the use of syncmers, we also test a standard hash-based sampling, providing an
235 unbiased estimate of Jaccard similarity as well. To sample with a given sampling rate $1/\nu$,
236 hash-based sampling uses a random hash function $h : \Sigma^k \rightarrow [0.. \nu - 1]$ with good statistical
237 properties, and samples a k -mer q iff $h(q) = 0$.

238 Our approach consists in storing sampled k -mers in IBLTs and apply the IBLT-difference
239 technique to recover set differences. Then, Jaccard similarity is estimated by

$$240 \quad J(A, B) = \frac{|A| - |A \setminus B|}{|A| + |B \setminus A|} = \frac{|B| - |B \setminus A|}{|B| + |A \setminus B|}. \quad (3)$$

241 Note that cardinalities $|A|$ and $|B|$ can be easily retrieved from respective IBLTs T_A and T_B
242 by summing all counter values and dividing by r .

243 3.4 IBLT dimensioning with syncmers

244 Dimensioning an IBLT holding syncmers requires estimating the expected number of dif-
245 ferences in the set difference of involved k -mer sets. Assuming that input datasets are
246 close genomic sequences of size L related by a mutation rate bounded by p_m and that k is
247 sufficiently large so that k -mer occurrences are unique, we can estimate the set difference.
248 Each mutation results in $2k$ k -mers in the set difference (k k -mers on each side), and therefore
249 the size of set difference is estimated to be $2kp_m L$. Taking into account density $\frac{2}{k-z+1}$ of
250 syncmers (Section 2.2), we obtain the estimation

$$251 \quad n = \frac{4kLp_m}{k - z + 1}. \quad (4)$$

252 3.5 Approximating k -mer set differences

253 The method of Section 3.3 allows estimating Jaccard similarity on k -mers by Jaccard similarity
254 on syncmers. Here we describe how we can extend these ideas in order to recover *all* k -mers
255 from $K(S_1) \setminus K(S_2)$ and $K(S_2) \setminus K(S_1)$, where S_1, S_2 are input datasets and $K(S)$ denotes
256 the set of k -mers of a dataset S .

257 Note first that a straightforward way of doing this, through IBLTs of $K(S_1)$ and $K(S_2)$,
258 requires a considerable space because a single mutation generates a difference of k k -mers.

259 Using syncmers, we can “pack” k -mers into longer strings, compute the differences and then
260 recover k -mers from them. The set of recovered k -mers, however, will be a superset of exact
261 differences.

262 To achieve this, instead of storing syncmers, we store in IBLTs *extended* syncmers of
263 length $2k - z$. Extended syncmers are obtained by extending each syncmer to the right by
264 $k - z$ bases. Since successive syncmers overlap by at least z bases, this ensures that each
265 k -mer belongs to at least one extended syncmer.

266 By applying the IBLT-difference technique (Section 3.3), we obtain the extended syncmers
267 that differ between the two datasets, from which we extract k -mers and discard those shared
268 between the two obtained sets. It may still happen that the sets we obtain are *supersets*
269 of exact differences, due to the fact that an extended syncmer can contain a k -mer which
270 belongs to another extended syncmer common to both datasets. However, we state that
271 for a sufficiently large k , the fraction of common k -mers in those sets will be small enough,
272 which we illustrate experimentally in Section 4.5.

273 4 Results

274 To validate our ideas, we performed experiments on simulated sequences as well as on two
275 real-life datasets:

- 276 ■ covid: subsample of 50 *SARS-CoV-2* genomes¹. Sequence names are provided in Table
277 2.
- 278 ■ spneu: subsample of 28 *Streptococcus Pneumoniae* genomes from [5] whose names are
279 reported in Table 3. The subsample has been chosen to contain very close strains, with
280 pairwise mutation rates between them not exceeding 0.0005.

281 4.1 Comparison of different sampling approaches

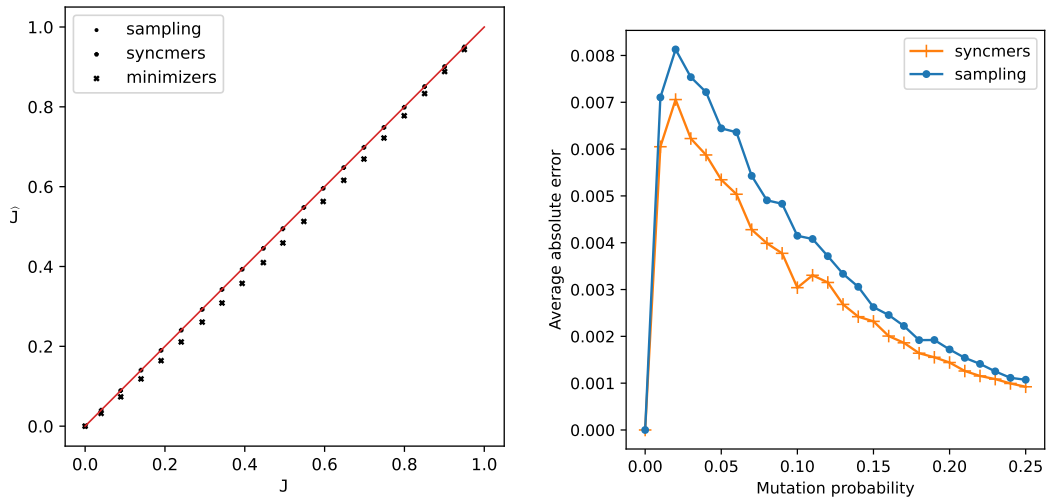
282 Random sampling, minimizers and syncmers have been compared by computing Jaccard
283 similarities between pairs of synthetic sequences. Each pair is constructed by first generating a
284 uniform random sequence of length L and then mutating it through independent substitutions.
285 Points in Figure 1a are averages over $T = 500$ independent trials. For fairness of comparison,
286 parameters for uniform sampling, minimizers and syncmers have been chosen to guarantee
287 the same sampling rate $1/\nu$. We know that $c_s \approx 2/(k - z + 1)$, $c_m \approx 2/(w + 1)$ and $c_{ns} \approx 1/\nu$
288 are the densities of syncmers, minimizers and random sampling, respectively. Thus, given
289 parameters k and z , setting the minimizer window length as $w = k - z$ and choosing a
290 sampling rate $1/\nu = c_s$ ensures about the same number of sampled k -mers for all algorithms.
291 As Figure 1a shows, syncmers do not have the previously reported biased behaviour of
292 minimizers [1], but they seem to be comparable to random sampling. However, as shown in
293 Figure 1b, random sampling is subject to larger errors than syncmers, due to less uniform
294 distribution along the sequence. For these reasons, we choose syncmer sampling as the mean
295 to reduce IBLT memory in Section 4.3.

296 4.2 Space performance of IBLTs

297 In order to demonstrate the space efficiency of IBLTs in our framework, we compare them
298 against a solution based on KMC k -mer counting software [17]. KMC provides an efficient

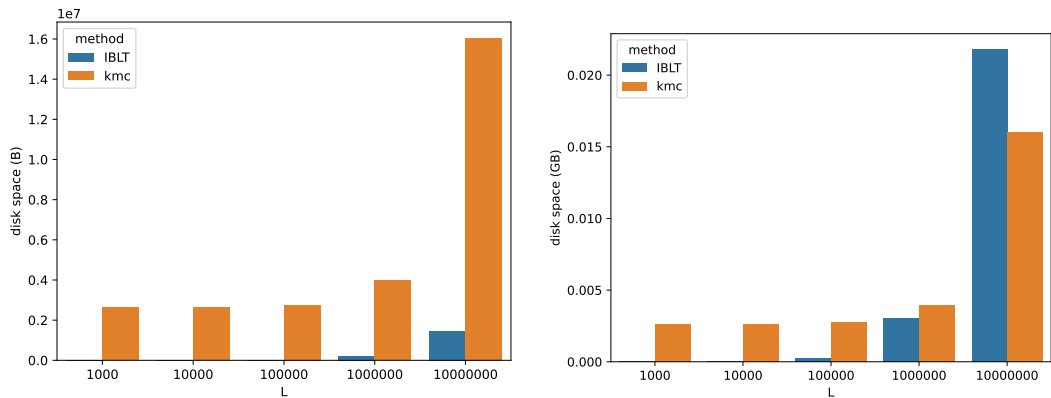
¹ <https://www.ncbi.nlm.nih.gov/datasets/coronavirus/genomes/>

14:8 Efficient reconciliation of genomic datasets of high similarity



(a) Minimizers present a non-negligible bias as opposed to syncmers and random sampling which are unbiased (and overlap in the plot). Each measurement was repeated 500 times on random sequences of length $L = 10\text{K}$ with $k = 15$, $w = 11$ (for minimizers) and $z = 4$ (for syncmers). Sampling rate is given by $1/\nu = 2/(k - z + 1) = 1/6$. **(b)** Absolute average errors for syncmers and random sampling, as a function of mutation rate. Parameters are unchanged from Figure 1a since both plots are generated from the same data.

■ **Figure 1** Comparison between random sampling, minimizers and syncmers



(a) $p_m = 0.001$

(b) $p_m = 0.01$

■ **Figure 2** Space taken by IBLTs depends on the similarity between stored sets. For very similar sequences (mutation rate $p_m = 0.001$, Figure 2a), IBLTs are more space-efficient than KMC. Their advantage appears reduced for increased p_m and large sequences (Figure 2b).

299 way for storing, manipulating and querying sets of k -mers. Unlike other counting tools
 300 (Jellyfish [21] or DSK [27]), KMC allows easy sorting of its output which leads to an efficient
 301 way to compute Jaccard similarity.

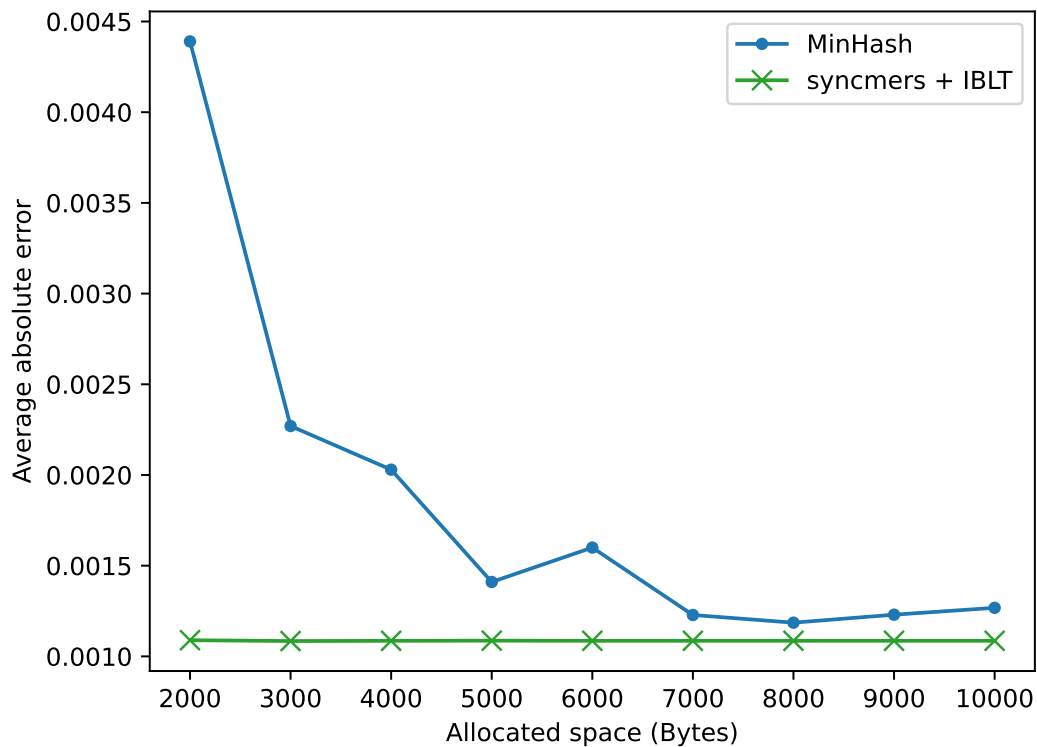
302 We compared memory taken by IBLTs vs. KMC databases for storing syncmers issued
 303 from two similar sequences. For this, we applied the same procedure as in Section 4.1:
 304 mutating a random sequence of length L with mutation probability p_m . Sampled syncmers
 305 from both sequences are stored respectively in IBLTs and KMC databases. Figure 2 reports

306 average space taken by the two data structures. Each bar is the average over $T = 100$ trials,
 307 except for case $L = 10M$ for which $T = 10$. IBLTs were dimensioned (see (1)) to guarantee
 308 peelability of all T sketches with high probability.

309 Figure 2a clearly demonstrates the advantage of IBLTs when the mutation rate is small.
 310 For larger p_m and long sequences, the number of differences reach a point where exact data
 311 structures become preferable, as illustrated by Figure 2b for $p_m = 0.01$ and sequences of
 312 length 10M.

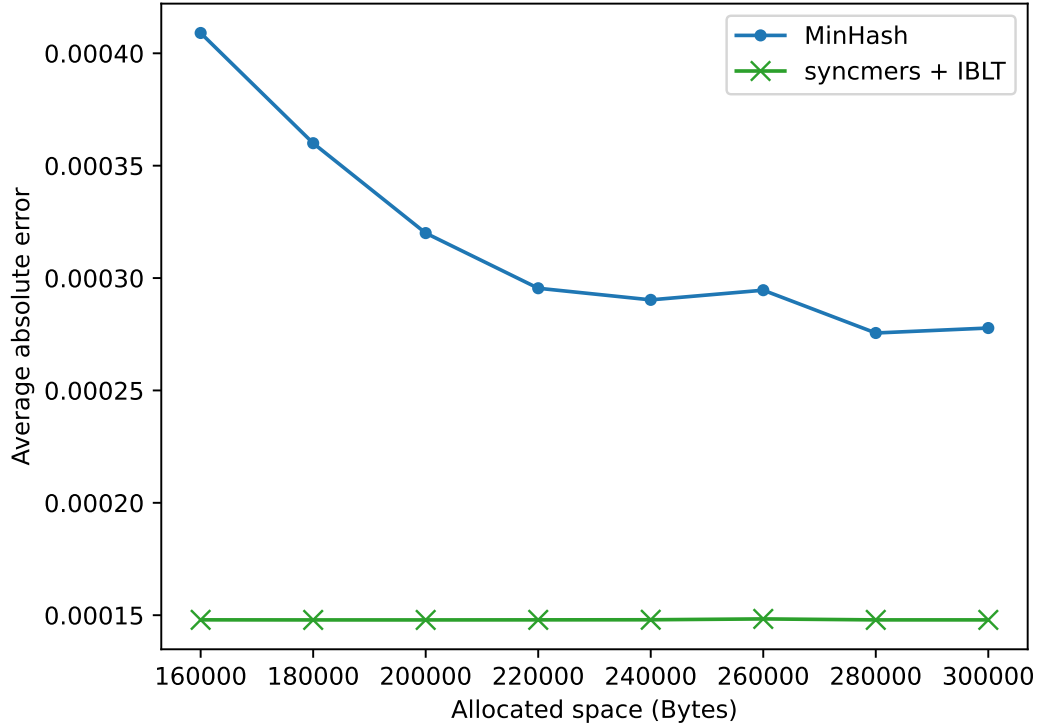
313 In our experiments, subtracting one IBLTs from another is dominated by the time taken
 314 to load/save the sketches, and not by performing the actual difference. Even in more complex
 315 scenarios, subtraction remains a very simple operation that can be performed by accessing
 316 one bucket at a time in any given order. On the other hand, the amount of time required by
 317 listing the content of an IBLT varies greatly and depends on the set of items stored in it.
 318 Slow-downs primarily happen when a new starting point is needed during peeling.

319 4.3 Accuracy of Jaccard similarity estimation from IBLTs of syncmers



■ **Figure 3** Comparison between IBLTs and MinHash for computing pairwise Jaccard on the `covid` dataset. The x-axis reports the amount of space allocated for each sketch while the y-axis reports the average absolute error. The number of hashes s for MinHash sketches and IBLTs sizes are chosen accordingly. $k = 15$ and $z = 4$ in all tests. Sketch size for MinHash and table size for IBLTs are chosen to fit the allocated memory.

320 Figures 3 and 4 report comparisons of both IBLTs and MinHash sketches on `covid`
 321 and `spneu` datasets respectively. Both plots show the average absolute error of Jaccard
 322 estimate computed over all pairs of sequences of the respective dataset. Exact Jaccard
 323 similarities computed over the full k -mer sets are used as ground truth. MinHash sketches



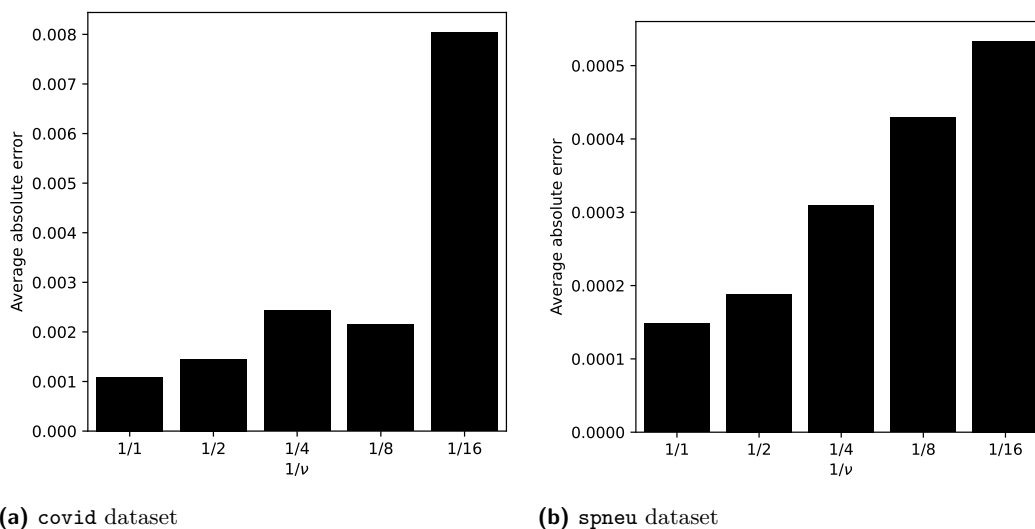
■ **Figure 4** Comparison between IBLTs and MinHash for computing pairwise Jaccard on the `spneu` dataset with the same setting as Figure 3.

324 (line MINHASH in the plots) were implemented using MASH [24]. All sketch sizes (in bytes)
 325 are fixed beforehand with both MinHash sketches and IBLTs dimensioned accordingly in
 326 order to fit the allocated memory. The number of bits allocated for payload field P in our
 327 IBLT implementation is set to be the minimum multiple of 8 larger than or equal to $2k$. As
 328 MASH [24] uses 32- or 64-bit hashes, we used $k = 15$ in our experiments in order to force
 329 both methods to use 32-bit representations.

330 In all experiments, IBLTs storing syncmers (line SYNCMERS + IBLT) showed the best
 331 precision. For covid genomes (Figure 3), full MinHash sketches become competitive for larger
 332 sketch sizes. Unlike MinHash, the average error of IBLTs remains constant across all reported
 333 cases because over-dimensioning only increases the probability of successful listing. For the
 334 `spneu` dataset (Figure 4), MinHash errors are about twice those of IBLTs across all allocated
 335 sketch sizes confirming that IBLTs are more memory-efficient. The general conclusion is that
 336 if sequences to be compared are highly similar, IBLTs storing syncmers are more efficient
 337 than MinHash sketches, with the latter being better suited to quickly provide an overview
 338 over more diverging datasets.

339 4.4 Sampling syncmers for further space reductions

340 Since syncmer sampling rate ($\frac{2}{k-z+1}$) cannot be made arbitrarily small for a given k , we
 341 also tested the effect of additional downstream sampling of syncmers, before inserting them
 342 into IBLTs. To this end, Figure 5 reports a comparison of syncmers sampled with different
 343 sampling rates $1/\nu$. We observe that downstream sampling of syncmers comes at the cost of
 344 decreased precision for both datasets (Figure 5a and 5b), but it might be useful to further



■ **Figure 5** Effect of sampling syncmers before IBLT insertion on the average absolute error. $1/\nu$ is the compression rate used for sampling syncmer sets before IBLT insertion. $\nu = 1$ means no sampling (full syncmer sets).

345 reduce space.

346 4.5 Experiments on approximating k -mer set differences

347 We tested the method from Section 3.5 of approximating k -mer set differences on both the
 348 covid dataset and on two random datasets. Each random dataset contains 50 sequences of
 349 length 30000 obtained by first generating a uniform random sequence which is then mutated
 350 49 times using a mutation probability p_m .

351 Recall that the method of Section 3.5 allows one to compute a *superset* of the symmetric
 352 difference $(K(S_1) \setminus K(S_2)) \cup (K(S_2) \setminus K(S_1))$ of sets of k -mers occurring in datasets S_1 and
 353 S_2 . Here we measure the precision of this method, that is the number of spurious k -mers
 354 found by the algorithm. Those are k -mers actually belonging to $K(S_1) \cap K(S_2)$ but output
 355 by the algorithm as if they belong to $(K(S_1) \setminus K(S_2)) \cup (K(S_2) \setminus K(S_1))$.

356 Table 1 summarizes the experiments. Columns ‘diff’ and ‘err’ show the average/max-
 357 imum cardinality of the true set difference and spurious k -mers, respectively, over all pairs
 358 of sequences. In the case of random datasets, sequences were generated with mutation
 359 probabilities $p_m = 0.01$ and $p_m = 0.001$.

	covid		random			
	diff	err	$p_m = 0.001$		$p_m = 0.01$	
			diff	err	diff	err
average	325.29	11.03	1708.78	60.03	15342.81	357.57
max	661	31	2396	110	17047	486

■ **Table 1** True size of symmetric difference of k -mer sets and its overestimate. For each experiment, ‘diff’ is the average/maximum size of the true symmetric difference, and ‘err’ is the average/maximum number of spurious k -mers reported as being in the symmetric difference. p_m is the mutation probability used to generate sequences from a random one.

360 We observe that the number of spurious k -mers remains small, on average within about
 361 3% of the true set difference size.

362 **5** Conclusions

363 To the best of our knowledge, our work is the first to apply Invertible Bloom Lookup Tables
 364 to k -mer processing for alignment-free comparison of DNA sequence datasets. We showed
 365 that whenever involved datasets are similar enough and their similarity can be bounded *a*
 366 *priori*, IBLTs lead to a more space-efficient and, at the same time, more accurate method
 367 for estimating Jaccard similarity of underlying k -mer sets. This is achieved by combining
 368 IBLTs with k -mer sampling via syncmers. As opposed to minimizers, syncmers provide
 369 an unbiased estimator of Jaccard index, which was confirmed in our experiments. At the
 370 same time, syncmer sampling is shown to lead to a more concentrated estimator than the
 371 straightforward hash-based sampling. Thus, IBLTs combined with syncmers constitute a
 372 powerful alternative to MinHashing for estimating Jaccard similarity for similar datasets.
 373 Note that in the context of viral/bacterial pan-genomics, dealing with similar datasets is a
 374 predominant situation in bioinformatics. In particular, the number of closely related bacterial
 375 and viral strains is rapidly growing.

376 As another application of IBLTs, we are able to approximately compute differences of
 377 underlying k -mer sets in small space. This opens new prospects as k -mers proper to a
 378 dataset can be used to infer information about genetic variation, specific mutation, etc.
 379 Note that MINHASH is designed to only estimate similarity and is not capable of providing
 380 information about actual differences. We also believe that by using additional space-efficient
 381 data structures this method can be extended to compute *exact* set differences on more
 382 complex datasets and we plan to explore this in our future work.

383 Our ideas may have further useful applications, for example to reconciliation of datasets
 384 located on remote computers, in which case IBLTs could avoid transmitting entire datasets
 385 (similar to a scenario described in [12]). Another example is a selection of sufficiently diverse
 386 datasets avoiding redundancy, as e.g. [18]. Note finally that IBLTs may also act as filters
 387 for filtering out dissimilar datasets: in this case, non-peelability of the difference IBLT is an
 388 indicator of dissimilarity.

389 References

- 390 1 Mahdi Belbasi, Antonio Blanca, Robert S. Harris, David Koslicki, and Paul Med-
 391 vedev. The minimizer Jaccard estimator is biased and inconsistent. *bioRxiv*,
 392 2022. arXiv:<https://www.biorxiv.org/content/early/2022/01/17/2022.01.14.476226>.
 393 full.pdf, doi:10.1101/2022.01.14.476226.
- 394 2 Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. COBS: A Compact
 395 Bit-Sliced Signature Index. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String*
 396 *Processing and Information Retrieval*, Lecture Notes in Computer Science, pages 285–303,
 397 Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-32686-9_21.
- 398 3 Phelim Bradley, Henk C Den Bakker, Eduardo P. C. Rocha, Gil McVean, and Zamin
 399 Iqbal. Ultra-fast search of all deposited bacterial and viral genomic data. *Nature biotechno-*
 400 *logy*, 37(2):152–159, February 2019. URL: [https://www.ncbi.nlm.nih.gov/pmc/articles/](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6420049/)
 401 [PMC6420049/](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6420049/), doi:10.1038/s41587-018-0010-1.
- 402 4 Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings.*
 403 *Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, June
 404 1997. doi:10.1109/SEQUEN.1997.666900.

- 405 5 Karel Brinda, Alanna Callendrello, Kevin C. Ma, Derek R. MacFadden, Themoula Char-
406 alampous, Robyn S. Lee, Lauren Cowley, Crista B. Wadsworth, Yonatan H. Grad, Gregory
407 Kucherov, Justin O’Grady, Michael Baym, and William P. Hanage. Rapid inference of
408 antibiotic resistance and susceptibility by genomic neighbour typing. *Nature Microbiology*,
409 5(3):455–464, March 2020. URL: <https://www.nature.com/articles/s41564-019-0656-6>,
410 doi:10.1038/s41564-019-0656-6.
- 411 6 Dan DeBlasio, Fiyinfoluwa Gbosibo, Carl Kingsford, and Guillaume Marçais. Practical
412 universal k-mer sets for minimizer schemes. In *Proceedings of the 10th ACM International*
413 *Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB ’19,
414 page 167–176, New York, NY, USA, 2019. Association for Computing Machinery. doi:
415 10.1145/3307339.3342144.
- 416 7 Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus
417 Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via xorsat. In *Proceedings of*
418 *the 37th International Colloquium Conference on Automata, Languages and Programming*,
419 ICALP’10, page 213–225, Berlin, Heidelberg, 2010. Springer-Verlag.
- 420 8 Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in
421 biological sequences. *PeerJ*, 9:e10805, February 2021. URL: [https://peerj.com/articles/](https://peerj.com/articles/10805)
422 10805, doi:10.7717/peerj.10805.
- 423 9 Barış Ekim, Bonnie Berger, and Yaron Orenstein. A Randomized Parallel Algorithm for
424 Efficiently Finding Near-Optimal Universal Hitting Sets. In Russell Schwartz, editor, *Research*
425 *in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 37–53, Cham,
426 2020. Springer International Publishing. doi:10.1007/978-3-030-45257-5_3.
- 427 10 David Eppstein and Michael T. Goodrich. Straggler identification in round-trip data streams
428 via Newton’s identities and invertible Bloom filters. *IEEE Transactions on Knowledge and*
429 *Data Engineering*, 23(2):297–306, 2011. doi:10.1109/TKDE.2010.132.
- 430 11 Huan Fan, Anthony R. Ives, Yann Surget-Groba, and Charles H. Cannon. An assembly and
431 alignment-free method of phylogeny reconstruction from next-generation sequencing data.
432 *BMC Genomics*, 16(1):522, July 2015. doi:10.1186/s12864-015-1647-5.
- 433 12 Michael T. Goodrich and Michael Mitzenmacher. Invertible Bloom lookup tables, 2011. URL:
434 <https://arxiv.org/abs/1101.2245>, doi:10.48550/ARXIV.1101.2245.
- 435 13 Gaurav Gupta, Minghao Yan, Benjamin Coleman, R. A. Leo Elworth, Todd Treangen, and
436 Anshumali Shrivastava. Sub-linear Sequence Search via a Repeated And Merged Bloom Filter
437 (RAMBO): Indexing 170 TB data in 14 hours. *arXiv:1910.04358 [cs, q-bio]*, December 2019.
438 arXiv: 1910.04358. URL: <http://arxiv.org/abs/1910.04358>.
- 439 14 Robert S Harris and Paul Medvedev. Improved representation of sequence Bloom trees.
440 *Bioinformatics*, 36(3):721–727, 08 2019. arXiv:[https://academic.oup.com/bioinformatics/](https://academic.oup.com/bioinformatics/article-pdf/36/3/721/38712473/btz662.pdf)
441 [article-pdf/36/3/721/38712473/btz662.pdf](https://academic.oup.com/bioinformatics/article-pdf/36/3/721/38712473/btz662.pdf), doi:10.1093/bioinformatics/btz662.
- 442 15 Chirag Jain, Luis M. Rodriguez-R, Adam M. Phillippy, Konstantinos T. Konstantinidis,
443 and Srinivas Aluru. High throughput ANI analysis of 90K prokaryotic genomes reveals
444 clear species boundaries. *Nature Communications*, 9(1):5114, November 2018. URL: [https:](https://www.nature.com/articles/s41467-018-07641-9)
445 [://www.nature.com/articles/s41467-018-07641-9](https://www.nature.com/articles/s41467-018-07641-9), doi:10.1038/s41467-018-07641-9.
- 446 16 Mikhail Karasikov, Harun Mustafa, Gunnar Rätsch, and André Kahles. Lossless indexing
447 with counting de bruijn graphs. *bioRxiv*, 2022. arXiv:[https://www.biorxiv.org/content/](https://www.biorxiv.org/content/early/2022/02/03/2021.11.09.467907.full.pdf)
448 [early/2022/02/03/2021.11.09.467907.full.pdf](https://www.biorxiv.org/content/early/2022/02/03/2021.11.09.467907.full.pdf), doi:10.1101/2021.11.09.467907.
- 449 17 Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and man-
450 ipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 05 2017. arXiv:[https://](https://academic.oup.com/bioinformatics/article-pdf/33/17/2759/25163903/btx304.pdf)
451 academic.oup.com/bioinformatics/article-pdf/33/17/2759/25163903/btx304.pdf, doi:
452 10.1093/bioinformatics/btx304.
- 453 18 Nathan LaPierre, Mohammed Alser, Eleazar Eskin, David Koslicki, and Serghei Mangul.
454 Metalign: efficient alignment-based metagenomic profiling via containment min hash. *Genome*
455 *Biology*, 21(1):242, September 2020. doi:10.1186/s13059-020-02159-0.

14:14 Efficient reconciliation of genomic datasets of high similarity

- 456 19 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–
457 3100, 05 2018. arXiv:[https://academic.oup.com/bioinformatics/article-pdf/34/18/](https://academic.oup.com/bioinformatics/article-pdf/34/18/3094/25731859/bty191.pdf)
458 [3094/25731859/bty191.pdf](https://academic.oup.com/bioinformatics/article-pdf/34/18/3094/25731859/bty191.pdf), doi:10.1093/bioinformatics/bty191.
- 459 20 Camille Marchet, Christina Boucher, Simon J. Puglisi, Paul Medvedev, Mikaël Salson, and
460 Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing
461 data sets. *Genome Research*, 31(1):1–12, January 2021. URL: [https://www.ncbi.nlm.nih.](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7849385/)
462 [gov/pmc/articles/PMC7849385/](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7849385/), doi:10.1101/gr.260604.119.
- 463 21 Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting
464 of occurrences of k-mers. *Bioinformatics*, 27(6):764, March 2011. URL: [https://www.ncbi.](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3051319/)
465 [nlm.nih.gov/pmc/articles/PMC3051319/](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3051319/), doi:10.1093/bioinformatics/btr011.
- 466 22 Michael Molloy. The pure literal rule threshold and cores in random hypergraphs. In
467 *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04,
468 pages 672–681, USA, January 2004. Society for Industrial and Applied Mathematics.
- 469 23 Martin D Muggli, Bahar Alipanahi, and Christina Boucher. Building large updatable colored
470 de Bruijn graphs via merging. *Bioinformatics*, 35(14):i51–i60, 07 2019. arXiv:[https://](https://academic.oup.com/bioinformatics/article-pdf/35/14/i51/28913259/btz350.pdf)
471 academic.oup.com/bioinformatics/article-pdf/35/14/i51/28913259/btz350.pdf, doi:
472 10.1093/bioinformatics/btz350.
- 473 24 Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman,
474 Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation
475 using MinHash. *Genome Biology*, 17(1):132, June 2016. doi:10.1186/s13059-016-0997-x.
- 476 25 Giulio Ermanno Pibiri. Sparse and skew hashing of k-mers. *bioRxiv*, 2022. arXiv:
477 <https://www.biorxiv.org/content/early/2022/01/18/2022.01.15.476199.full.pdf>,
478 doi:10.1101/2022.01.15.476199.
- 479 26 Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting.
480 In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching*, pages 173–182,
481 Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 482 27 Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. DSK: k-mer counting with very
483 low memory usage, March 2013. URL: <https://pubmed.ncbi.nlm.nih.gov/23325618/>, doi:
484 10.1093/bioinformatics/btt020.
- 485 28 Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke.
486 Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–
487 3369, December 2004. doi:10.1093/bioinformatics/bth408.
- 488 29 Kamil Salikhov, Gustavo Sacomoto, and Gregory Kucherov. Using cascading Bloom filters
489 to improve the memory usage for de Bruijn graphs. *BMC Algorithms for Molecular Biology*,
490 9(1):2, 2014. URL: <http://www.almob.org/content/9/1/2>.
- 491 30 Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for
492 document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on*
493 *Management of data*, SIGMOD '03, pages 76–85, San Diego, California, June 2003. Association
494 for Computing Machinery. doi:10.1145/872757.872770.

495 **6 Appendix**496 **Datasets**■ **Table 2** Names of covid genomes used for Figure 3

BS001151.1	LR877722.1	LR883214.1	MT520216.1
MT706180.1	MT757082.1	MT800758.1	MT834020.1
MT970159.1	MT971010.1	MT973151.1	MW064390.1
MW064919.1	MW064981.1	MW153809.1	MW153954.1
MW154711.1	MW156712.1	MW184416.1	MW184648.1
MW190904.1	MW190957.1	MW191020.1	MW191146.1
MW206148.1	MW276931.1	MW321243.1	MW321430.1
MW593629.1	MW631874.1	MW669599.1	MW681303.1
MW681489.1	MW693959.1	MW696216.1	MW702101.1
MW708072.1	MW708184.1	MW708826.1	MW720341.1
MW733722.1	MW738615.1	MW749542.1	MW776764.1
MW820211.1	MW850083.1	MW863243.1	MW868532.1
MW868533.1	MW871079.1		

■ **Table 3** Names of *S.Pneumoniae* genomes used for Figure 4

BZ2I7.fa	R34-3087.fa	007649.fa	R34-3097.fa
4PYM0.fa	JBYFY.fa	T8Z8O.fa	R34-3044.fa
O61U7.fa	81LMX.fa	O0RHB.fa	R34-3083.fa
R34-3025.fa	WAMFH.fa	O811E.fa	R34-3164.fa
CCV1H.fa	0U64I.fa	6893Z.fa	1VDX8.fa
R34-3074.fa	R34-3227.fa	LS3OB.fa	UTEDZ.fa
REAOU.fa	R34-3229.fa	067094.fa	4K4C9.fa