Efficient reconciliation of genomic datasets of high similarity

Yoshihiro Shibuya
LIGM, Universitè' 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, Univ. Gustave Eiffel, Marne-la-Vallée, France

Abstract

We apply Invertible Bloom Lookup Tables (IBLTs) to 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 take 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 a 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.CVIT.2016.23

Supplementary Material https://github.com/yhhshb/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 [19, 12, 15]. To deal with this issue, a considerable effort has been put to developing optimized data structures, with succinct solutions [23, 21, 15] and approximate membership data structures [26, 12, 2, 13, 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 [22] 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 unbiased estimator of the Jaccard similarity index [4]. A MinHash sketch can thus be viewed as a sample of the set of k-mers of the sequence it represents. Given that s is much smaller

© Yoshihiro Shibuya and Djamal Belazzougui and Gregory Kucherov;
licensed under Creative Commons License CC-BY 4.0
42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

than the genome length, working with the sampled hashes leads to fast pairwise comparisons using small memory. However, when two sequences are close and share most of their k-mers, MinHash sketches of small size are not able to reliably estimate their degree of similarity.

In this work, we propose an alternative approach to evaluate the difference in k-mer composition of two related datasets. Our method relies on Invertible Bloom Lookup Table (IBLT) data structure [11, 10] which is an extension of Bloom filters, supporting deletions of items and, most importantly, enumeration (with high probability) of stored items. One of the applications of IBLT is reconciliation of two sets of items: in a scenario considered in [11], a set A is stored in an IBLT which is then transmitted to the holder of another set B. By screening B against the IBLT of A it is possible to recover the items $A \setminus B$ and $B \setminus A$, with high probability. This is done through the so-called *peeling* procedure [7].

In this paper we make one step further: inspired by ideas of [24], we recover both $A \setminus B$ and $B \setminus A$ from IBLTs of A and B, rather than from an IBLT of one of them and the whole other set. Furthermore, a crucial property is that the *size of these IBLTs is proportional to the symmetric difference size* $A \setminus B \cup B \setminus A$ rather than to the size of the original sets. This provides a key to the efficiency of our solution when input sets are similar: even if input sets are very big, their difference can be recovered using a data structure (sketch) whose size is proportional to the size of the difference of those sets rather than of the sets themselves. Estimating the symmetric difference allows us to estimate the Jaccard similarity, using information about the sizes of input sets. Thus, whereas close datasets require larger MinHash sketches to be properly compared, our method, on the contrary, requires smaller memory.

Another ingredient of our solution is k-mer sampling. Intuitively, since two adjacent k-mers share $k-1$ bases, the information stored in the set of all k-mers appears highly redundant. One popular method of sampling k-mers from genomic sequences is based on minimizers [25]. Under this technique, consecutive sampled k-mers are within a bounded distance from each other and therefore no large portion of the sequence can remain unsampled. Another favorable property is that similar regions are likely to yield similar samples of minimizers. However, it has recently been shown that estimating Jaccard similarity based on minimizer sampling leads to a bias [1]. Here we propose to replace minimizers by syncmers [8]. Syncmers provide another way of k-mer sampling which has certain advantages over minimizers. As opposed to minimizers, syncmers are not context-dependent: for a k-mer to be a syncmer depends on the k-mer alone regardless the context where it occurs. As a consequence, under reasonable probabilistic assumptions, syncmer sampling leads to an unbiased estimate of Jaccard similarity, as the fraction of syncmers among shared k-mers (intersection) is expected to be the same as that among all k-mers (union). We experimentally show that this is indeed the case.

By combining syncmer sampling with IBLTs, we obtain a space-efficient method for accurately estimating Jaccard similarity for similar datasets. For datasets of high similarity, the proposed method is superior to the popular MinHash algorithm [22], both in terms of memory and precision. We also propose an application of this technique to retrieving k-mers that differ between given two datasets. Our method computes a superset of those k-mers with a limited number of spurious k-mers. In particular, under assumption that each k-mer occurs once, our method computes the exact set differences between involved k-mer sets. We validate our algorithms on both simulated data and on real datasets made of *SARS-CoV-2* and *Straphilococcus Pneumoniae* genomes.

2 Technical preliminaries

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

2.1 Minimizers

Independently introduced in [25] and [27], minimizers are defined by a triplet of parameters (k, w, h), where k is the k-mer length, w a window size, and h a function defining an order on k-mers. h is usually chosen to be an appropriately defined hash function, the lexicographical order is rarely used in practice due to its poor statistical properties.

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

Since two neighboring windows at positions i and $i + 1$ are likely to share their minimizer, minimizers provide a way to sample k-mers from a sequence with bounded distance between consecutive sampled k-mers. An advantage of this sampling strategy is that similar sequences will likely have similar lists of minimizers, which makes it useful for mapping algorithms [18, 14]. Under reasonable assumptions, the density of minimizers, i.e. fraction of sampled k-mers, is $\frac{2}{w+1}$ [25, 8]. If minimizer positions in the original sequence are not important, they can be discarded and the resulting k-mer multiset can be reduced to a simple k-mer set.

2.2 Syncmers

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

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

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

2.3 Invertible Bloom Lookup Tables

Invertible Bloom Lookup Tables (IBLT) [10, 11] are a generalization of Bloom filters for storing a set of elements (keys), drawn from a large universe, possibly associated with attribute values. In contrast to Bloom filters, in addition to insertion of elements, IBLTs

support deletion of elements as well as listing all elements stored in the data structure. The latter operation succeeds with high probability (w.h.p.) depending on the number of stored elements relative to the size of the data structure. An important property is that this probability depends only on the number of elements stored at the moment of listing, and not across the entire lifespan of the data structure. Thus, at a given time, an IBLT can store a number of elements greatly exceeding the threshold for which it was built, returning to be fully functional whenever a sufficiently number of deletions has taken place.

An IBLT is an array T of m buckets together with r (pairwise independent) hash functions mapping U to $[0..m-1]$ and an additional global hash function h_e on U. Each bucket i contains three fields: a counter C, a key field P and a hash field H, where C counts the number of keys hashed to bucket i, P stores the XOR-sum of the keys hashed to bucket i, and H contains the XOR-sum of hash values of keys hashing to bucket i using h_e.

Adding an element p to the IBLT is done as follows. For each $j \in 1, \cdots, r$, we perform $T[i].C = T[i].C + 1$, $T[i].P = T[i].P \oplus p$, and $T[i].H = T[i].H \oplus h_e(p)$, where $\oplus$ stands for XOR. Given that XOR is the inverse operation of itself, deletion of p is done similarly except that $T[i].C = T[i].C - 1$.

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

At each moment, an IBLT is associated to a r-hypergraph where nodes are buckets and edges correspond stored elements with each edge including the buckets an element is hashed to. Listing the elements contained into an IBLT then relies on the peelability property of random hypergraphs [7, 20]. It is well-known that a 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 peelability threshold (for example, $c_3 \approx 1.23$ [11]). Thus, allocating

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

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

2.4 MinHash sketching

MinHash sketching was introduced in [4] as a method to estimate Jaccard similarity between two sets, applied to document comparison. In bioinformatics, MinHash was first applied in Mash software [22] and then successfully used in a number of other tools. Assume we are given a universe U and an order on U defined via a hash function h. For a set $A \subset U$, the bottom-s MinHash sketch of A, denoted $\mathbb{S}(A)$, is the set of s minimal elements of A (or their hashes), where s is a user-defined parameter. The Jaccard similarity index between two sets A and B, $J(A, B) = |A \cap B|/|A \cup B|$, can then be estimated from the sketches of A and B, namely

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

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

The Jaccard similarity between k-mer sets of two dataset constitutes a biologically relevant measure of their similarity. In particular, if involved datasets are genomic sequences, this measure allows one to estimate the mutation rate between the sequences [22].

3 Methods

3.1 Set reconciliation from two IBLTs

Invertible Bloom Lookup Tables can be used to achieve set reconciliation between two sets A and B, that is to recover sets $A \setminus B$ and $B \setminus A$. Under a scenario described in [11], the holder of A stores it in an IBLT T_A which is then transmitted to the holder of B. Elements of B are then deleted from T_A. In the resulting IBLT, P-fields with $T_A[i].C = 1$ correspond to elements of $A \setminus B$ and those with $T_A[i].C = -1$ to $B \setminus A$. The peeling process is applied to either of such fields. Whenever $T_A[i].C = 1$, we delete $p = T_A[i].P$ from T_A on condition 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 on condition that $h_e(p) = T_A[i].H$. The process lists all elements of both $A \setminus B$ and $B \setminus A$ w.h.p.

Inspired by work [24], we modify the above scheme in order to recover the symmetric difference between A and B from their respective IBLTs T_A and T_B, rather than from the IBLT of one set and the whole other set. To do this this, assume that T_A and T_B are of the same size and use the same hash functions. We then compute the difference of T_A and T_B, denoted T_{A-B} and defined through $T_{A-B}[i].C = T_A[i].C - T_B[i].C$, $T_{A-B}[i].P = T_A[i].P \oplus T_B[i].P$, 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" in T_{A-B}, that is, T_{A-B} holds elements of $A \setminus B \cup B \setminus A$ with elements from $A \setminus B$ occurring "positively" and those from $B \setminus A$ occurring "negatively". The peeling process described above results then in listing the elements of both $A \setminus B$ and $B \setminus A$.

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

3.2 Making buckets lighter

In the above scheme of IBLT difference, the H field becomes important as the case $T_{A-B}[i].C = 1$ (or $T_{A-B}[i].C = -1$) can occur due to a spurious "cancelling out" of distinct elements. However, to save space, we propose to get rid of the H field and replace the "checksum" verification by another test: if $T_{A-B}[i].C = 1$ (resp. $T_{A-B}[i].C = -1$), we check whether $p = T_{A-B}[i].P$ is a valid element by checking if $h_j(p) = i$ for one of $j \in [1..r]$. This allows us to save space at the price of additional verification time. This technique works particularly well for large IBLTs but it becomes less effective for small ones, as the "false positive" probability is proportional to the size of the table.

3.3 Combining sampling and IBLTs for Jaccard similarity estimation

We now turn to our main goal: estimating Jaccard similarity of two k-mers sets using IBLTs. The common approach uses MinHash sketching as described in [22] (see Section 2.4). However, MinHash requires larger sketches to measure similarity of close datasets. One possible idea could be to store MinHash sketches in IBLTs in hope to use them for estimating Jaccard

similarity through the IBLT-difference scheme from the previous section. This, however, runs into an obstacle due to the fact that applying (2) requires knowledge of k-mers belonging to the sketch intersection, and not only to sketch differences.

Rather than working with the entire sets of k-mers, we resort to sampling. It is known that sampling minimizers incurs a bias in estimating Jaccard similarity [1]. We propose to use syncmers instead of minimizers which don't suffer from being context-dependent and provide then an unbiased estimator of Jaccard similarity.

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

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

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

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

3.4 IBLT dimensioning with syncmers

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

$$n = \frac{4kLp_m}{k - z + 1}. \tag{4}$$

3.5 Approximating k-mer set differences

The method of Section 3.3 allows estimating Jaccard similarity on k-mers by Jaccard similarity on syncmers. Here we describe how we can extend these ideas in order to recover *all* k-mers 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 the set of k-mers of a dataset S.

Note first that a straightworward way of doing this, through IBLPs of $K(S_1)$ and $K(S_2)$, requires a considerable space because a single mutation generates a difference of k k-mers. Using syncmers, we can "pack" k-mers into longer strings, compute the differences and then recover k-mers from them. The set of recovered k-mers, however, will be a superset of exact differences.

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

By applying the IBLT-difference technique (Section 3.3), we obtain the extended syncmers that differ between the two datasets, from which we extract k-mers and discard those shared

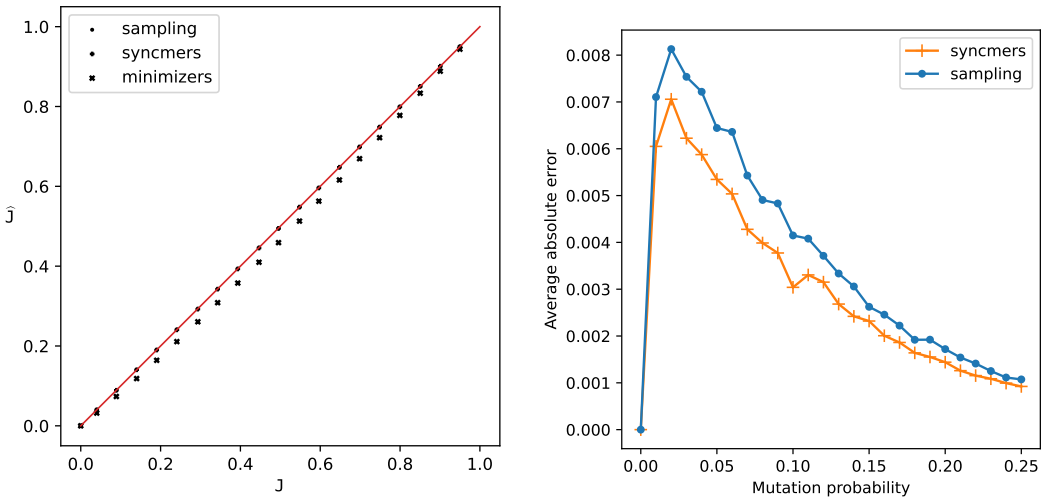
between the two obtained sets. It may still happen that the sets we obtain are *supersets* of exact differences, due to the fact that an extended syncmer can contain a k-mer which belongs to another extended syncmer common to both datasets. However, we state that for a sufficiently large k, the fraction of common k-mers in those sets will be small enough, which we illustrate experimentally in Section 4.4. In the extreme case when each k-mer occurs once, our method computes exact k-mer set differences.

4 Results

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

- `covid`: subsample of 50 *SARS-CoV-2* genomes[1]. Sequence names are provided in Table 2.
- `spneu`: subsample of 28 *Streptococcus Pneumoniae* from [5] whose names are reported in Table 3. All sequences are guaranteed to have mutation rates below 0.0005.

4.1 Comparison of different sampling approaches



(a) Minimizers present a non-negligeable bias as opposed to syncmers and random samples which are unbiased. Each measurement was repeated 500 times on random sequences of length $L = 10$K with $k = 15$, $w = 11$ (for minimizers) and $z = 4$ (for syncmers)

(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.

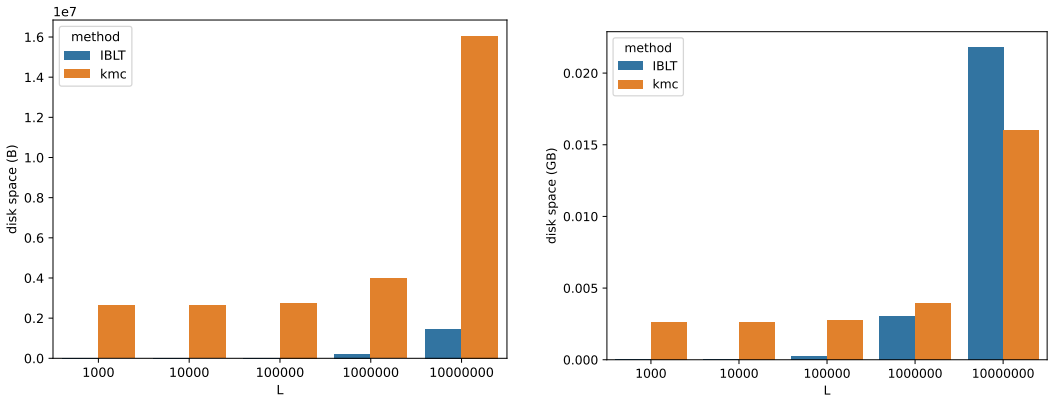
Figure 1 Comparison between random sampling, minimizers and syncmers

Random sampling, minimizers and syncmers have been compared by computing Jaccard similarities between pairs of synthetic sequences. Each pair is constructed by first generating a uniform random sequence of length L and then by mutating it. Points in Figure 1a are averages over $T = 500$ independent trials. For fairness of comparison, parameters for uniform sampling, minimizers and syncmers have been chosen to guarantee the same sampling rate

[1] `https://www.ncbi.nlm.nih.gov/datasets/coronavirus/genomes/`

(density). We know that $c_s = 2/(k - z + 1)$, $c_m = 2/(w + 1)$ and $c_{ns} = 1/\nu$ are the densities of syncmers, minimizers and random sampling, respectively. Thus, given parameters k and z, setting the minimizer window length as $w = k - z$ and choosing a sampling rate $\nu = c_s$ ensures about the same number of sampled k-mers for all algorithms. As Figure 1a shows, syncmers do not have the previously reported biased behaviour of minimizers [1], but they seem to be comparable to random sampling. However, as shown in Figure 1b, random sampling is subject to larger errors than syncmers, due to much less uniform distribution along the sequence. For these reasons, we choose to store syncmers inside IBLTs in section 4.3.

4.2 Space performance of IBLTs



(a) $p_m = 0.001$ **(b)** $p_m = 0.01$

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

To demonstrate the performance of IBLTs in our framework, we chose to compare them to a solution based on KMC software [16]. Original and mutated sequences of each pair are first sampled into sets of syncmers. These sets are then stored into IBLTs and KMC databases. As in the previous section, tests are performed on simulated data.

Figure 2 presents space comparisons between IBLTs and KMC databases. Each bar is the average over $T = 100$ trials for all sequence lengths, except case $L = 10$M for which $T = 10$. All sketches are insured to be peelable for all reported trials by setting the over-dimensioning constant ε accordingly. IBLTs were dimensioned by using formula (4), where the Lp_m was replaced by the true number of substitutions introduced by the mutation process. Figure 2a clearly demonstrates the advantage of IBLTs when the mutation rate is small. Figure 2b shows that for larger p_m, IBLTs need more space to guarantee peelability. If the number of expected differences is high, IBLTs can be less efficient than exact data structures, as for $p_m = 0.01$ and sequences of length 10M, see Figure 2b. Inserting a new element into an IBLT takes less than 1μs. In our experiments, subtracting one IBLTs from another is dominated by the time taken to load/save the sketches, and not by performing the actual difference. Even in more complex scenarios, subtraction remains a very simple operation that can be performed by accessing one bucket at a time in any given order. On the other hand, the amount of time required by listing the content of an IBLT varies greatly and depends on the set of items stored in it. For an easily peelable sketch, listing is as fast as insertion. However,

slow-downs are possible when multiple passes are needed to find new starting points for peeling.

4.3 Accuracy of Jaccard similarity estimation from IBLTs of syncmers

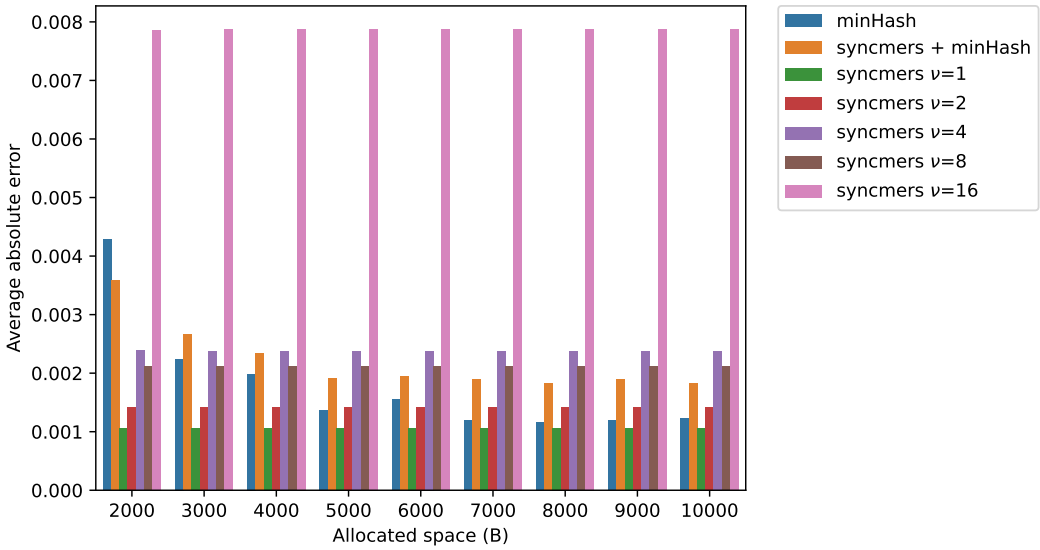


Figure 3 Comparison between IBLTs and MinHash for computing pairwise Jaccard on the `covid` dataset. The x-axis reports the amount of space allocated for each sketch while the y-axis reports the average absolute error. $k = 15$ and $z = 4$ in all tests. Sketch size is kept constant for all methods, with the number of hashes s for MinHash sketches and IBLTs sizes chosen accordingly. ν is the sampling rate used for sampling syncmer sets before IBLT insertion. $\nu = 1$ means no sampling (full syncmer sets).

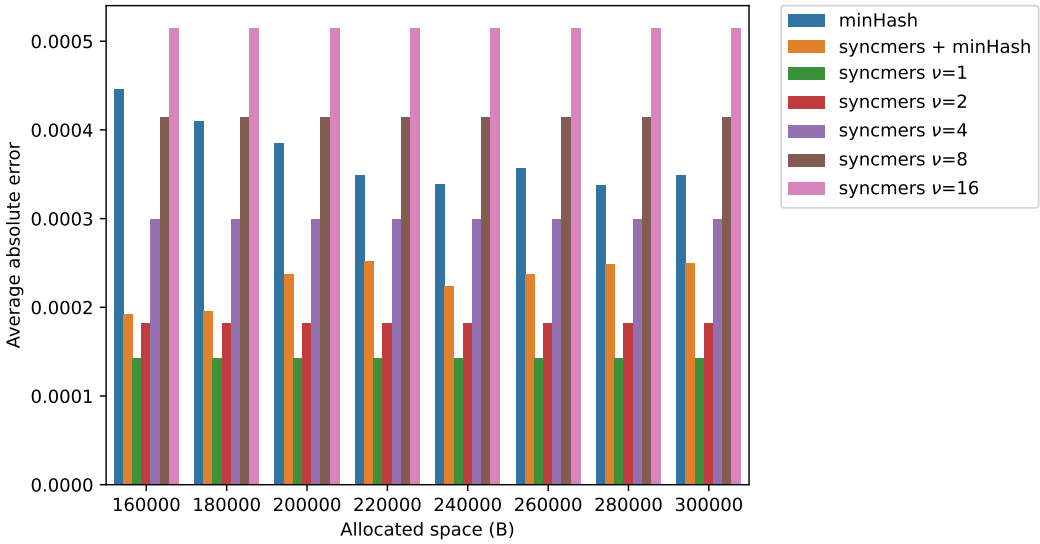


Figure 4 Accuracy of IBLTs and MinHash for computing pairwise Jaccard on the `spneu` dataset with the setting as Figure 3.

Figures 3 and 4 report the average absolute error when computing Jaccard similarities using both IBLTs and MinHash (Mash) sketches on `covid` and `spneu` datasets respectively. MinHash sketches are built either on the full set of k-mers (column MinHash), or on the set of selected syncmers used by IBLTs (column syncmers + MinHash).

Since syncmer sampling rate ($\frac{2}{k-z+1}$) cannot be made arbitrarily small, we also test the effect of additional downstream sampling of syncmers (before inserting them into IBLTs). To this end, Figure 3 also reports cases of syncmers sampled with different sampling rates $1/\nu$ (columns syncmers $\nu = \cdots$). All sketch sizes (in bytes) are fixed beforehand with both MinHash sketches and IBLTs dimensioned accordingly in order to satisfy this constraint.

In all experiments, IBLTs storing all syncmers ($\nu = 1$) showed the best precision. For covid genomes (Figure 3), full MinHash skeches become competitive for larger sketch sizes. Unlike MinHash, the average error of IBLTs for a given sampling rate remains constant across all reported cases because over-dimensioning only increases the probability of successful listing. Errors of MinHash sketches built over syncmers closely follow the same pattern as MinHash built on full k-mer sets. However, for the covid dataset and larger sketch sizes (Figure 3), they perform worse. Sampling syncmers with compression factors $\nu > 1$ comes at the cost of decreased precision, but might be useful to further reduce IBLTs space. For the `spneu` dataset (Figure 4), MinHash errors are about twice those of IBLTs across all allocated sketch sizes confirming that IBLTs are more memory-efficient. As before, storing full syncmer sets ($\nu = 1$) inside IBLTs leads to the smallest errors, whereas larger sampling rates might help in reducing space at the cost of worse estimations.

The general conclusion is that if sequences to be compared are highly similar, IBLTs storing syncmers (with or without sampling) can be more efficient than MinHash sketches, with the latter being better suited to quickly provide an overview over more heterogeneous datasets.

4.4 Experiments on approximating k-mer set differences

We tested the method of approximating k-mer set differences (Section 3.5) on both the `covid` dataset and on random data. The two random dataset each contains 50 sequences of length 30000 built by first generating a uniform random sequence, which is then mutated 49 times using mutation probabilities $p_m = 0.01$ and $p_m = 0.001$. Table 1 summarizes the results.

	`covid`		random			
			$p_m = 0.001$		$p_m = 0.01$	
	diff	err	diff	err	diff	err
average	318.79	10.81	1674.60	58.83	15035.95	350.30
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 size of the true symmetric difference, and 'err' is the average number of additional (spurious) k-mers obtained.

The results illustrate that the number of spurious k-mers in the symmetric difference (column "err" of Table 1) is only a small fraction of the size of the whole true difference computed over k-mers (column "diff").

5 Conclusions

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

As another application of IBLTs, we are able to approximately compute differences of underlying k-mer sets using a small space. This opens new prospects as k-mers proper to a dataset can be used to infer information about genetic variation, specific mutation, etc. We also believe that using additional space-efficient data structures this method can be extended to compute *exact* set differences and plan to explore this in our future work.

Our ideas may have further useful applications, for example to reconciliation of datasets located on remote computers, in which case IBLTs could avoid transmitting entire datasets (similar to a scenario described in [11]). Another example is a selection of sufficiently diverse datasets avoiding redundancy, as e.g. [17]. Note finally that IBLTs may also act as filters for filtering out dissimilar datasets: in this case, non-peelability of the difference IBLT is an indicator of dissimilarity.

References

1. Mahdi Belbasi, Antonio Blanca, Robert S. Harris, David Koslicki, and Paul Medvedev. The minimizer jaccard estimator is biased and inconsistent. *bioRxiv*, 2022. URL: `https://www.biorxiv.org/content/early/2022/01/17/2022.01.14.476226`, `arXiv:https://www.biorxiv.org/content/early/2022/01/17/2022.01.14.476226.full.pdf`, `doi:10.1101/2022.01.14.476226`.
2. Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. COBS: A Compact Bit-Sliced Signature Index. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval*, Lecture Notes in Computer Science, pages 285–303, Cham, 2019. Springer International Publishing. `doi:10.1007/978-3-030-32686-9_21`.
3. Phelim Bradley, Henk C Den Bakker, Eduardo P. C. Rocha, Gil McVean, and Zamin Iqbal. Ultra-fast search of all deposited bacterial and viral genomic data. *Nature biotechnology*, 37(2):152–159, February 2019. URL: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6420049/`, `doi:10.1038/s41587-018-0010-1`.
4. A. Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, June 1997. `doi:10.1109/SEQUEN.1997.666900`.
5. Karel Břinda, Alanna Callendrello, Kevin C. Ma, Derek R. MacFadden, Themoula Charalampous, Robyn S. Lee, Lauren Cowley, Crista B. Wadsworth, Yonatan H. Grad, Gregory Kucherov, Justin O'Grady, Michael Baym, and William P. Hanage. Rapid inference of antibiotic resistance and susceptibility by genomic neighbour typing. *Nature Microbiology*, 5(3):455–464, March 2020. URL: `https://www.nature.com/articles/s41564-019-0656-6`, `doi:10.1038/s41564-019-0656-6`.

6 Dan DeBlasio, Fiyinfoluwa Gbosibo, Carl Kingsford, and Guillaume Marçais. Practical universal k-mer sets for minimizer schemes. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '19, page 167–176, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3307339.3342144.

7 Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via xorsat. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming*, ICALP'10, page 213–225, Berlin, Heidelberg, 2010. Springer-Verlag.

8 Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, February 2021. URL: https://peerj.com/articles/10805, doi:10.7717/peerj.10805.

9 Barış Ekim, Bonnie Berger, and Yaron Orenstein. A Randomized Parallel Algorithm for Efficiently Finding Near-Optimal Universal Hitting Sets. In Russell Schwartz, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 37–53, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45257-5_3.

10 David Eppstein and Michael T. Goodrich. Straggler identification in round-trip data streams via Newton's identities and invertible Bloom filters. *IEEE Transactions on Knowledge and Data Engineering*, 23(2):297–306, 2011. doi:10.1109/TKDE.2010.132.

11 Michael T. Goodrich and Michael Mitzenmacher. Invertible Bloom lookup tables, 2011. URL: https://arxiv.org/abs/1101.2245, doi:10.48550/ARXIV.1101.2245.

12 Gaurav Gupta, Minghao Yan, Benjamin Coleman, R. A. Leo Elworth, Todd Treangen, and Anshumali Shrivastava. Sub-linear Sequence Search via a Repeated And Merged Bloom Filter (RAMBO): Indexing 170 TB data in 14 hours. *arXiv:1910.04358 [cs, q-bio]*, December 2019. arXiv: 1910.04358. URL: http://arxiv.org/abs/1910.04358.

13 Robert S Harris and Paul Medvedev. Improved representation of sequence Bloom trees. *Bioinformatics*, 36(3):721–727, 08 2019. arXiv:https://academic.oup.com/bioinformatics/article-pdf/36/3/721/38712473/btz662.pdf, doi:10.1093/bioinformatics/btz662.

14 Chirag Jain, Luis M. Rodriguez-R, Adam M. Phillippy, Konstantinos T. Konstantinidis, and Srinivas Aluru. High throughput ANI analysis of 90K prokaryotic genomes reveals clear species boundaries. *Nature Communications*, 9(1):5114, November 2018. URL: https://www.nature.com/articles/s41467-018-07641-9, doi:10.1038/s41467-018-07641-9.

15 Mikhail Karasikov, Harun Mustafa, Gunnar Rätsch, and André Kahles. Lossless indexing with counting de bruijn graphs. *bioRxiv*, 2022. URL: https://www.biorxiv.org/content/early/2022/02/03/2021.11.09.467907, arXiv:https://www.biorxiv.org/content/early/2022/02/03/2021.11.09.467907.full.pdf, doi:10.1101/2021.11.09.467907.

16 Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 05 2017. arXiv:https://academic.oup.com/bioinformatics/article-pdf/33/17/2759/25163903/btx304.pdf, doi:10.1093/bioinformatics/btx304.

17 Nathan LaPierre, Mohammed Alser, Eleazar Eskin, David Koslicki, and Serghei Mangul. Metalign: efficient alignment-based metagenomic profiling via containment min hash. *Genome Biology*, 21(1):242, September 2020. doi:10.1186/s13059-020-02159-0.

18 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 05 2018. arXiv:https://academic.oup.com/bioinformatics/article-pdf/34/18/3094/25731859/bty191.pdf, doi:10.1093/bioinformatics/bty191.

19 Camille Marchet, Christina Boucher, Simon J. Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, January 2021. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7849385/, doi:10.1101/gr.260604.119.

20 Michael Molloy. The pure literal rule threshold and cores in random hypergraphs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, pages 672–681, USA, January 2004. Society for Industrial and Applied Mathematics.

21 Martin D Muggli, Bahar Alipanahi, and Christina Boucher. Building large updatable colored de Bruijn graphs via merging. *Bioinformatics*, 35(14):i51–i60, 07 2019. `arXiv:https://academic.oup.com/bioinformatics/article-pdf/35/14/i51/28913259/btz350.pdf`, `doi:10.1093/bioinformatics/btz350`.

22 Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17(1):132, June 2016. `doi:10.1186/s13059-016-0997-x`.

23 Giulio Ermanno Pibiri. Sparse and skew hashing of k-mers. *bioRxiv*, 2022. URL: `https://www.biorxiv.org/content/early/2022/01/18/2022.01.15.476199`, `arXiv:https://www.biorxiv.org/content/early/2022/01/18/2022.01.15.476199.full.pdf`, `doi:10.1101/2022.01.15.476199`.

24 Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching*, pages 173–182, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

25 Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, December 2004. `doi:10.1093/bioinformatics/bth408`.

26 K. Salikhov, G. Sacomoto, and G. Kucherov. Using cascading Bloom filters to improve the memory usage for de Brujin graphs. *BMC Algorithms for Molecular Biology*, 9(1):2, 2014. URL: `http://www.almob.org/content/9/1/2`.

27 Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 76–85, San Diego, California, June 2003. Association for Computing Machinery. `doi:10.1145/872757.872770`.

6 Appendix

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	O8I1E.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