Set-Min sketch: a probabilistic map for power-law distributions with application to k-mer annotation

Yoshihiro Shibuya, Djamal Belazzougui, Gregory Kucherov

Set-Min sketch: a probabilistic map for power-law distributions with application to k-mer annotation

Yoshihiro Shibuya[1, *], Djamal Belazzougui[2], and Gregory Kucherov[1,3]

[1]LIGM, CNRS, Univ. Gustave Eiffel, Marne-la-Vallée, France

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

[3]Skolkovo Institute of Science and Technology, Moscow, Russia

[*]To whom correspondence should be addressed

October 26, 2021

Keywords: sketching, Set-Min sketch, Max-Min sketch, k-mer counting, k-mer spectrum, power-law distribution

Abstract: k-mer counts are important features used by many bioinformatics pipelines. Existing k-mer counting methods focus on optimizing either time or memory usage, producing in output very large count tables explicitly representing k-mers together with their counts. Storing k-mers is not needed if the set of k-mers is known,

making it possible to only keep counters and their association to k-mers. Solutions avoiding explicit representation of k-mers include Minimal Perfect Hash Functions (MPHFs) and Count-Min sketches. We introduce Set-Min sketch – a sketching technique for representing associative maps inspired from Count-Min – and apply it to the problem of representing k-mer count tables. Set-Min is provably more accurate than both Count-Min and Max-Min – an improved variant of Count-Min for static datasets that we define here. We show that Set-Min sketch provides a very low error rate, both in terms of the probability and the size of errors, at the expense of a very moderate memory increase. On the other hand, Set-Min sketches are shown to take up to an order of magnitude less space than MPHF-based solutions, for fully assembled genomes and large k. Space-efficiency of Set-Min in this case takes advantage of the power-law distribution of k-mer counts in genomic datasets.

Availability: `https://github.com/yhhshb/fress`

1 Introduction

Counting k-mer occurrences in genomic sequences is a rather common task in many bioinformatics pipelines. It is often the first step performed before a more complicated analysis, and its main applications go from read trimming (Liu *et al.*, 2012) to alignment-free variant calling (Rahman *et al.*, 2018; Khorsand and Hormozdiari, 2019). In recent years, many k-mer counting algorithms have been proposed, such as Jellyfish (Marçais and Kingsford, 2011), DSK (Rizk *et al.*, 2013) or KMC (Kokot *et al.*, 2017). Each of these methods was conceived with a particular trade-off between speed and main memory consumption. Jellyfish primarily addresses speed, whereas DSK and KMC primarily target memory

efficiency. All these tools output a map associating k-mers to their counts. Such a map can require a fairly big amount of disk space, especially when large values of k are used. For example, the KMC output for a human genome with $k = 32$ weights in at around 28 GB. Memory efficiency remains an important issue even when outputs are compressed. One way to partially solve this issue is to only store counters, without storing k-mers themselves. This idea is supported by the fact that, in many applications, we only deal with k-mers that come from partially assembled reads or succinct representations of k-mer sets such as colored de Bruijn graphs (Holley and Melsted, 2019), or spectrum-preserving string sets (Břinda *et al.*, 2020; Rahman and Medvedev, 2020), that is, only k-mers present in the original data are queried for their frequencies.

The idea of storing only counter information and not k-mers is also supported by the observation that the number of distinct k-mer counts in genomic data is relatively small. Furthermore, it is known that k-mer counts in fully assembled genomes obey a "heavy-tail" power-law distribution[1] with a relatively large absolute value of the exponent (Csűrös *et al.*, 2007; Chor *et al.*, 2009). For such distributions, the number of *distinct* k-mers makes a linear fraction of the data size, while the number of *distinct* k-mer counts is relatively small. For example, for a human genome and $k = 27$, there are about 2.5 billions distinct k-mers but only about 8,000 distinct frequency values. A heavy-tail distribution also imply that the majority of k-mers have a very small count: in the above example, 97% of k-mers are unique and 99% of k-mers have a count of at most 5. Frequent k-mers often tend to have an identical count as well, due to transposable elements: for example, k-mers specific to Alu repeats in primate genomes will likely have the same count.

Our contribution. We propose a new probabilistic data structure that we call *Set-Min sketch* capable to represent k-mer counts information in a small

[1]Here we assume k to be sufficiently large, typically $k > \log_4 L$, where L is the data size.

space and with small errors. The sketch guarantees that the expected *cumulative error* obtained when querying all k-mers of the dataset can be bounded by εN where N is the number of *all* k-mers (i.e. essentially, the size of the dataset). We provide a theoretical analysis in order to dimension the sketch according to the desired error bound. Set-Min sketch is a general data structure in that it can be used to efficiently represent a mapping of k-mers to any type of labels. Nevertheless, it performs best when data is skewed and the number of possible labels is relatively small.

Applications. The problem considered in this paper is the probabilistic compression of k-mer count tables computed from fully assembled genomes, with large k values. A Set-Min sketch is able to provide a more space-efficient map than other representations with low errors when count distributions are very skewed. Set-Min sketch can have different uses.

An application of Set-Min sketches not explored here, is to act as a temporary representation while building more complex structures based on counters. Consider, for example, the exact computation of weighted pairwise distances between all pairs of genomes in a given set. Examples of such distances are the Bray-Curtis similarity measure, see e.g. (Benoit *et al.*, 2016), or Weighted Jaccard similarity estimation (Chum *et al.*, 2008). The most naive algorithm is to first produce count tables for each dataset and then compare them pairwisely to produce the desired output. Instead of storing whole tables, one can store multiple Set-Min sketches together with a presence-absence data structure, such as a Bloom filter. By doing so, the weighted comparison computation is reduced to a single pass through the presence-absence data structure with the counters of a given k-mer retrieved on-demand from the Set-Min sketches of the datasets in which the k-mer is found.

Another application is the quick sharing of counter information between

different computational units in a distributed setting. Consider a situation where both a server and one of its clients have access to the same genomic representation (say, a set of contigs). Only the server can efficiently perform k-mer counting, while the client has to process incoming data based on the counts, but doen't have the computational resources to perform counting by itself. In this case, the server could send a Set-Min sketch to the other node.

One further feature of Set-Min sketches is their mergeability in case of redundant maps. In this scenario, a large map can be split into m sub-maps without the restriction of having disjoint sets of keys. Even if some maps have redundant information, i.e. share common (key,value) pairs, the Set-Min sketch built by cell-wise union of the m sketches will be equivalent to the sketch built from the whole original map. Note that Count-Min sketches are not mergeable for redundant maps but are mergeable when constituent maps have to be "added up". Set-Min sketches don't have this property. In this respect, Set-Min and Count-Min sketches may have complementary uses.

Outline of the paper. In Section 2, we start by formally introducing the relevant terminology and underlying methods. Section 3 presents the idea and algorithmic foundations of our method. Our results are presented in Section 4 while Section 5 discuss the limitations of our method and possible workarounds. We conclude in Section 5 with a summary of the work.

2 Background

2.1 k-mer spectrum

A k-mer spectrum is a distribution of k-mer frequencies across all k-mers occurring in the data, showing how many k-mers support each frequency value. For large values of k, k-mer spectra follow a power-law distribution (Csűrös

et al., 2007; Chor *et al.*, 2009) characterized by a linear-like dependence when represented in log-log scale. That is, the k-mer frequency distribution fits a dependence $f(t) \approx c \cdot t^{-a}$, where a is usually greater than 2. An example of the spectrum for the human reference genome with k equal to 32 is given in Figure 1. According to this distribution, a very large fraction of k-mers have very low frequencies, while a few k-mers have "unexpectedly" large frequencies. Large value of a implies that there are relatively few distinct frequency values with non-zero support, whose number depends as the $\frac{1}{a}$-power of the number of k-mers.

2.2 Count-Min and Max-Min sketches

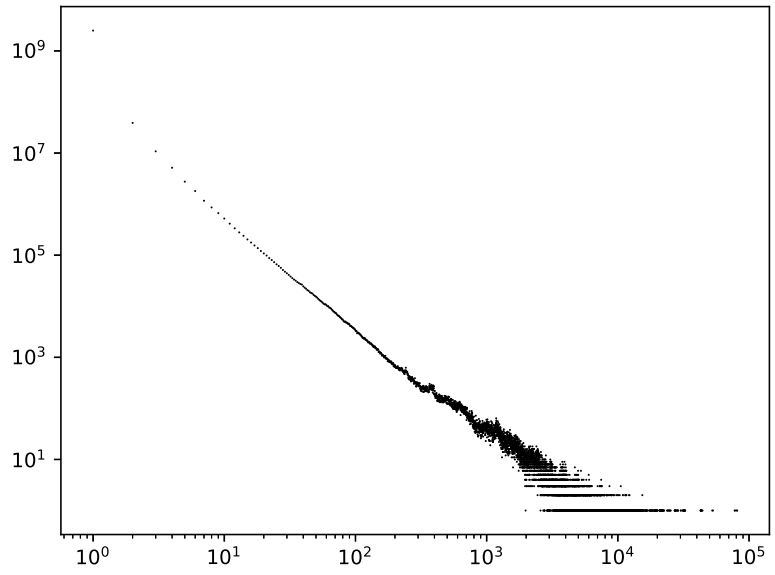

Figure 1: k-mer spectrum of the human genome for $k = 32$ in log-log scale.

Count-Min sketch (Cormode and Muthukrishnan, 2005a) is a method to

compactly represent an associative array $\boldsymbol{a}$ of counters in an approximated way. Count-Min is especially suitable for the streaming framework, when counters associated to keys can be updated dynamically. A Count-Min sketch is an $R \times B$ matrix A of counters where each row i is associated with a hash function $h_i(\cdot)$. To store counters and support their dynamic updates given in a stream, Count-Min works as follows. To process an update which is a (key,value) pair (p, ℓ), we perform $A(i, h_i(p)) = A(i, h_i(p)) + \ell$ for each row i. The (approximate) current counter associated with a key p is retrieved as $\hat{\boldsymbol{a}}(p) = \min_i\{h_i(p)\}$.

It has been shown that for a Count-Min sketch built on an associative array (vector) $\boldsymbol{a}$, with $R = \lceil \ln(\frac{1}{\delta}) \rceil$ and $B = \lceil \frac{e}{\varepsilon} \rceil$ for any given ε and δ, the over-estimate error of an individual counter is bounded by $\varepsilon \|\boldsymbol{a}\|_1$ with probability at least $1-\delta$, where $\|\boldsymbol{a}\|_1$ is the L_1-norm of $\boldsymbol{a}$ (Cormode and Muthukrishnan, 2005a). If counts follow a Zipfian rank-frequency distribution with parameter $b > 1$, B can be reduced to $O(\varepsilon^{-1/b})$ to guarantee the same bounds (Cormode and Muthukrishnan, 2005b). Note that $b > 1$ corresponds to $1 < a < 2$ in the corresponding spectrum distribution (Adamic, 2000), while k-mer spectra often fit a distribution with $a \geq 2$. Count-Min sketch supports negative updates, i.e. allows $\ell < 0$ in updates (p, ℓ), provided that the cumulative value for each key stays positive. If updates are only positive, there exists a modification of Count-Min leading to a better accuracy, mentioned in (Cormode, 2009) (therein attributed to (Estan and Varghese, 2002)) as *conservative update*. Under this modification, updates for each row i are made according to $A(i, h_i(p)) = \max\{A(i, h_i(p)), \hat{\boldsymbol{a}}(p) + \ell\}$, where $\hat{\boldsymbol{a}}(p)$ is the current Count-Min estimate of $\boldsymbol{a}(p)$. It is easily seen that under this scheme, $\hat{\boldsymbol{a}}(p)$ can still only over-estimate $\boldsymbol{a}(p)$, but cannot be larger than $\hat{\boldsymbol{a}}(p)$ computed by the original Count-Min. In this paper, we will deal with the static case when the value of any key is given once and never changes after that. In this framework, the conservative update strategy can be further modified by

defining updates as $A(i, h_i(p)) = \max\{A(i, h_i(p)), \ell\}$. We call this variant of the sketch Max-Min. In the static case, Max-Min sketch improves Count-Min without any computational overhead: it simply replaces addition by max in the update rule. As with the conservative update, one can check that estimates by Max-Min can only be over-estimates which, however, don't exceed estimates by original Count-Min.

2.3 Minimal Perfect Hashing

Minimal Perfect Hash Functions (MPHFs) are bijective functions between keys of a set S and integers in the range $[0, |S| - 1]$. By using hash values as indexes for an external array, it is possible to associate any type of information to the k-mers. The construction of MPHFs can be hyper-graph peeling-based (Yu *et al.*, 2017; Esposito *et al.*, 2019) or array-based (Müller *et al.*, 2014). The first family of algorithms leads to smaller MPHFs, close to theoretical space lower bound of 1.44 bits per key, while array-based MPHFs are conceptually simpler and have practical implementations for k-mer sets, such as BBHash (Limasset *et al.*, 2017).

3 Methods

3.1 Set-Min sketch in a nutshell

Assume we are given a set K of keys with associated values taken from a set L with $|L| \ll |K|$. In our case, K is the set of k-mers occurring in the dataset and L includes their frequencies, although our method will hold for any set of labels L. We want to compactly implement the associative map of (key,value) pairs. A Set-Min sketch is an $R \times B$ matrix M where each bucket is treated as a set, initially empty. Similar to the Count-Min sketch, rows in the matrix correspond

to hash functions h_i, $0 \leq i \leq R-1$, that we assume pairwise independent.

At construction time, the key of each (key,value) pair (p, ℓ) is hashed by the hash functions to retrieve its buckets and the value ℓ is inserted into each set. Formally, we update $M(i, h_i(p)) = M(i, h_i(p)) \cup \{\ell\}$ for each row i. To retrieve the value associated with a key p, we compute the intersection of the corresponding sets, that is $\cap_{0 \leq i \leq R-1} h_i(p)$. If the intersection is a singleton, the value is returned. If the intersection is empty, p is not present in the map. If the intersection contains more than one value, we have a collision.

3.2 Dealing with collisions

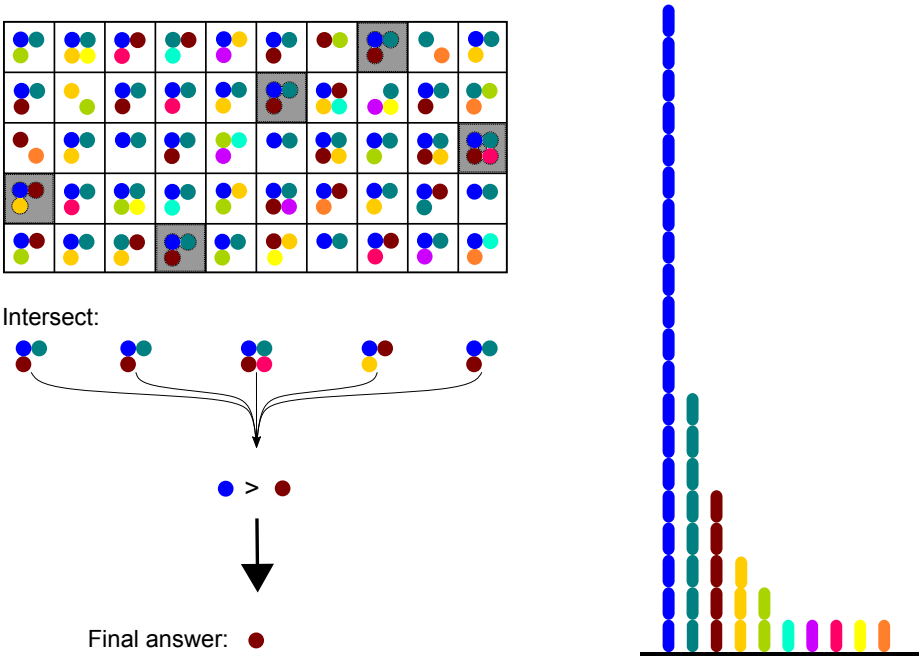

Figure 2: Example of collision resolution in case of multiple items occurring in the intersection. The brown label is returned because it is more rare compared to the blue one.

In case of collision, the choice is guided by the number of k-mers supporting each label of the intersection: the label with the smallest support is returned

(see Figure 2). The rationale for this is that the label with the smallest support has the smallest probability to appear "by chance", as labels with larger support occur in more buckets and are therefore more likely to occur in the intersection by chance. Thus, the algorithm compares spectrum values for all values of the intersection, and returns the label with the smallest value (ties are broken randomly). If the spectrum is monotonically decreasing (as it is usually the case for large k, see Figure 1), then the label returned is simply the largest one among those in the intersection.

In this work, we assume that only k-mers present in the dataset can be queried. In this case, a query can no longer result in an empty intersection. We further optimize by not storing in the sketch the label ℓ_1 with the largest support. For large k, ℓ_1 is usually 1 which is the frequency of the largest fraction of k-mers. ℓ_1 is retrieved implicitly: when the intersection is empty, ℓ_1 is returned. This optimization allows us to save space and will be further discussed later. Note that, with these modification, an error may occur even if the resulting intersection is a singleton but the right label is actually ℓ_1.

We now show that with Set-Min sketch, we can bound the *total* absolute error over all k-mers of the dataset. Consider a sketch S built on a map assigning to each k-mer $p \in K$ a value (label) $\ell_p \in L$ which is the frequency of p in the dataset. We denote by c_ℓ the number of k-mers with frequency $\ell \in L$ (spectrum value).

Consider

$$D = \sum_{p \in K} |\hat{\ell}_p - \ell_p| \tag{1}$$

where $\hat{\ell}_p$ is the label of p returned by the sketch. Our goal is to dimension R and B such that $D \leq \varepsilon \|\boldsymbol{a}\|_1$, where $\|\boldsymbol{a}\|_1$ is the total number of k-mers in the dataset (roughly, the dataset size) and $0 < \epsilon \leq 1$.

Querying p returns an incorrect frequency $m \neq \ell_p$ iff m occurs in the

intersection and $c_m < c_{\ell_p}$. The probability of this event is

$$\left(1 - \left(1 - \frac{1}{B}\right)^{c_m}\right)^R \approx \left(1 - e^{-\frac{c_m}{B}}\right)^R \tag{2}$$

and the expectation of the error when querying p is then

$$\sum_{\substack{c_m < c_{\ell_p} \\ m \in L}} |m - \ell_p| \left(1 - e^{-\frac{c_m}{B}}\right)^R. \tag{3}$$

Summing up over all k-mers, we obtain

$$\mathrm{E}[D] = \sum_{\ell \in L} c_\ell \sum_{\substack{c_m < c_\ell \\ m \in L}} |m - \ell| \left(1 - e^{-\frac{c_m}{B}}\right)^R. \tag{4}$$

The total number of k-mers is $\|\boldsymbol{a}\|_1 = \sum_{\ell \in L} \ell c_\ell$. Given $0 < \varepsilon \leq 1$, our goal is to choose B and R in order to ensure

$$\sum_{\ell \in L} c_\ell \sum_{\substack{c_m < c_\ell \\ m \in L}} |m - \ell| \left(1 - e^{-\frac{c_m}{B}}\right)^R < \varepsilon \sum_{\ell \in L} \ell c_\ell. \tag{5}$$

Assuming that k is sufficiently large and the spectrum is monotonically decreasing, i.e. $c_m < c_\ell$ iff $m > \ell$. (5) then rewrites to

$$\sum_{\ell \in L} c_\ell \sum_{\substack{m > \ell \\ m \in L}} (m - \ell) \left(1 - e^{-\frac{c_m}{B}}\right)^R < \varepsilon \sum_{\ell \geq 1} \ell c_\ell. \tag{6}$$

Assume now that the spectrum follows a power-law with large exponent, that is, $c_\ell = C \cdot \ell^{-a}$ for some $a > 2$. Note that under this assumption, the number of unique k-mers is $c_1 = C$, and the number of all k-mers is

$$\sum_{\ell \geq 1} \ell c_\ell = C \cdot \sum_{\ell \geq 1} \frac{1}{\ell^{a-1}} \leq C \cdot \frac{a-1}{a-2},$$

since $\zeta(s) = \sum_{i \geq 1} \frac{1}{i^s} \leq \frac{s}{s-1}$ for $s > 1$.

We then have the following result.

Theorem 1. *Given $0 < \varepsilon \leq 1$, if $B > C$ and R, B satisfy*

$$R \cdot \log \frac{B}{C} > \log \frac{1}{\varepsilon}, \tag{7}$$

then (6) holds.

Proof. Our goal is to estimate

$$\sum_{\ell \in L} c_\ell \sum_{\substack{m > \ell \\ m \in L}} (m - \ell) \left(1 - e^{-\frac{c_m}{B}}\right)^R, \tag{8}$$

where $c_\ell = C \cdot \ell^{-a}$. We assume $B > C$ and approximate $1 - e^{-\frac{c_m}{B}} \approx \frac{c_m}{B} = \frac{C}{B} m^{-a}$. We further lower-approximate (8) by replacing sums by integrals, thus obtaining

$$\int_1^\infty C \cdot \ell^{-a} \int_\ell^\infty (m - \ell) \left(\frac{C}{B} m^{-a}\right)^R dm \, d\ell. \tag{9}$$

Routine computation of the integral yields

$$C \left(\frac{C}{B}\right)^R \frac{1}{(aR - 2)(aR - 1)(aR + a + 3)}. \tag{10}$$

The inequality of the Theorem becomes

$$\left(\frac{C}{B}\right)^R \frac{1}{(aR - 2)(aR - 1)(aR + a + 3)} < \varepsilon \frac{a - 1}{a - 2}. \tag{11}$$

The Theorem follows. □

The theorem allows us to dimension the Set-Min sketch. For example, one can set $B = \alpha C$ for some constant $\alpha > 1$ and $R = \log_\alpha \frac{1}{\varepsilon}$.

Algorithm 1: Heuristic to compute R and B

Data: $\{c_\ell\}_{\ell \in L}$, $\|\boldsymbol{a}\|_1 = \sum_{\ell \in L} \ell c_\ell$, ε
Result: R and B
$R \leftarrow 1$;
$B \leftarrow 1.44 \times c_{max}$;
$T \leftarrow \varepsilon \|\boldsymbol{a}\|_1$;
$E \leftarrow \mathrm{E}(D)$ (computed by (4));
while $E > T$ **do**
    $R \leftarrow R + 1$;
    $E \leftarrow \mathrm{E}(D)$;
end
$M \leftarrow R \times B$;
while $E < T$ **do**
    $R \leftarrow R - 1$;
    $B \leftarrow \lceil \frac{M}{R} \rceil$;
    $E \leftarrow \mathrm{E}(D)$;
end
$R \leftarrow R + 1$;
$B \leftarrow \lceil \frac{M}{R} \rceil$;

3.3 Computing tighter sketch dimensions

Theorem 1 provides a way to dimension a Set-Min sketch, provided that the spectrum follows a power-law distribution with a sufficiently large parameter a. In order to validate these estimates experimentally, and, at the same time, obtain a tool for computing tighter values B and R for arbitrary spectra, we implemented a simple heuristic hill climbing algorithm to compute those values by directly solving equation 5.

Algorithm 1, given below, starts with $R = 1$ and some initial value of B and then iteratively increments R and recomputes (4) until equation (5) holds true. In the implementation, B is initially set to $1.44 \times c_{max}$, where c_{max} is the largest spectrum value. After such a value of R is found, the algorithm starts decrementing R while incrementing B to maintain the total space RB constant as long as (5) holds. The final R and B are thus the last which satisfied (5) in the decrementing loop. Note that, for both loops, there is a value of R for which

the exit condition is satisfied. The rationale for this step is to have as small R as possible in order to reduce the query time, while maintaining the total space.

4 Results

We implemented Set-Min in a software tool named `fress`, available at `https://github.com/yhhshb/fress`. The `fress` pipeline compares Set-Min with Count-Min and MPHF implementations. `BBHash` (Limasset *et al.*, 2017) (`https://github.com/rizkg/BBHash`) is the only external library required, as Count-Min is implemented within `fress`.

Rather than storing a set in each bucket of the sketch, `fress` only stores an index to an array of involved sets. Note that the current version of `fress` does not include any complex optimisation, such as multi-threading or bit-packing of the final matrix. Sorted spectra and lists of involved sets are explicitly stored in text format.

We tested Set-Min on six data sets of different size and complexity. Four of them are fully assembled genomes:

SAI Sakai strain of *Escherichia Coli* taken from (Yi and Jin, 2013) (NCBI accession number B000007),

MNO genome of *Drosophila melanogaster* from FlyBase[2],

RAI genome of *Gossypium Raimondii* (Hatje and Kollmar, 2012) downloadable from AFproject(Zielezinski *et al.*, 2019),

GRC human reference genome assembly GRCh38[3].

The other two contain unassembled reads:

[2] `http://flybase.org`

[3] `ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/405/GCA_000001405.15_GRCh38/seqs_for_alignment_pipelines.ucsc_ids/GCA_000001405.15_GRCh38_no_alt_analysis_set.fna.gz`

USAI Sakai strain at 5x coverage from AFproject,

SRR low-coverage human data SRR622461 from the 1000 Genomes Project[4]

Table 1 summarizes the characteristics of each data set for each value of k in our analysis. Observe that, while the number of distinct k-mers is comparable to the total number of k-mers (data size), the number of distinct k-mer counts is small. This is in accordance with the power-law distribution discussed in Section 2.1.

4.1 Set-Min vs Count-Min sketch

Table 2 compares Set-Min sketch to Count-Min sketch. Dimensions R and B were computed using Algorithm 1 to insure bound (5) to hold for $\varepsilon = 0.01$. Dimensions of Count-Min sketch were set to be the same. Value 1 is the most common count in all reported datasets and it was not inserted into Count-Min sketch in order to make the comparison againts Set-Min as fair as possible. Zero values are thus interpreted as the non-inserted count.

For ease of comparison, column T reports the threshold $\varepsilon\|\boldsymbol{a}\|_1$ given to Algorithm 1. Columns E_s and E_c report the actual total sum of errors for Set-Min and Count-Min, respectively. In all reported cases $E_s < T$, as expected. The total error of Count-Min, E_c is, most of the time, one order of magnitude larger than E_s. For SRR with $k = 15$, it even exceeds the total number of k-mers $\|\boldsymbol{a}\|_1$ in the dataset.

The average error of Set-Min is, in most cases, very close to 1, which suggests that the overwhelming majority of collisions occur between successive counts such as 1 and 2 – the most abundant ones in the spectra considered here. The average error of Count-Min is bigger but of the same order of magnitude, except for small k and unassembled datasets.

[4] `ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR622/SRR622461/SRR622461_1.fastq.gz` . Only the file SRR622461_1 is used in this study.

Table 1: Data sheet for the data sets used in our study. Columns T_k and D_k report the total number (in millions) of k-mers and the number of distinct k-mers, respectively. D_c reports the number of distinct k-mer counts. C_k reports the number (in millions) of distinct k-mers with a count value different from the most common one (which is 1 in all reported cases).

Type	Name	T_k [M]	k	D_k [M]	D_c	C_k [M]
assembled	SAI	5.50	11	2.38	69	1.30
			15	5.23	42	0.16
			21	5.30	28	0.11
	MNO	143	15	101	883	15.51
			21	122	710	4.10
			27	124	605	4.00
			32	125	522	4.01
	RAI	727	15	251	1944	105.71
			21	546	1019	57.13
			27	604	699	45.21
			32	632	540	38.59
	GRC	2935	15	547	12718	370.16
			21	2327	10038	95.15
			27	2483	7946	73.99
			32	2567	6651	66.71
unassembled	USAI	25	11	3.06	239	2.76
			15	9.74	143	6.74
			21	10.0	97	6.52
	SRR	7500	15	676	22323	538.56
			21	3635	17211	1435.30
			27	3734	13157	1353.95
			32	3703	10643	1261.06

On the other hand, the fraction of k-mers producing an error is in striking contrast: in case of Set-Min, about only 1-3% of distinct k-mers produce an error, while for Count-Min, this fraction is much larger. This shows that Count-Min cannot be used when most of k-mer counts are expected to be retrieved precisely, for comparable sketch sizes.

Table 2: Set-Min compared to Count-Min. T is the reference upper bound on the sum of errors equal to $\varepsilon\|\boldsymbol{a}\|_1$ (right-hand side of (5)). E_s and E_c are the sum of errors for Set-Min and Count-Min respectively. N_s and N_c are the percentages (rounded to integers) of distinct k-mers producing an error, for Set-Min and Count-Min, respectively. A_s and A_c are respective average errors, with average taken over the number of distinct k-mers resulting in an error in the respective sketch.

Name	k	R	B	T	E_s	E_c	N_s	N_c	A_s	A_c
SAI	11	4	$1.04 \cdot 10^6$	$5.50 \cdot 10^4$	$5.00 \cdot 10^4$	$1.39 \cdot 10^6$	1.8	26	1.15	2.24
SAI	15	5	$2.13 \cdot 10^5$	$5.50 \cdot 10^4$	$4.66 \cdot 10^4$	$2.38 \cdot 10^5$	0.9	4	1.01	1.1
SAI	21	5	$1.20 \cdot 10^5$	$5.50 \cdot 10^4$	$5.29 \cdot 10^4$	$4.57 \cdot 10^5$	0.9	7	1.05	1.18
USAI	11	5	$4.58 \cdot 10^5$	$2.57 \cdot 10^5$	$1.99 \cdot 10^5$	$7.44 \cdot 10^7$	2.6	99	2.46	24.6
USAI	15	4	$4.79 \cdot 10^6$	$2.49 \cdot 10^5$	$2.45 \cdot 10^5$	$8.01 \cdot 10^6$	1.9	33	1.31	2.53
USAI	21	5	$3.99 \cdot 10^6$	$2.38 \cdot 10^5$	$2.00 \cdot 10^5$	$7.91 \cdot 10^6$	1.6	34	1.21	2.35
MNO	15	5	$1.79 \cdot 10^7$	$1.43 \cdot 10^6$	$1.39 \cdot 10^6$	$8.35 \cdot 10^6$	1.4	7	1	1.25
MNO	21	4	$4.69 \cdot 10^6$	$1.43 \cdot 10^6$	$1.41 \cdot 10^6$	$2.26 \cdot 10^7$	1.1	12	1.03	1.61
MNO	27	5	$3.72 \cdot 10^6$	$1.43 \cdot 10^6$	$1.11 \cdot 10^6$	$2.28 \cdot 10^7$	0.9	12	1.01	1.48
MNO	32	5	$3.81 \cdot 10^6$	$1.42 \cdot 10^6$	$1.11 \cdot 10^6$	$2.10 \cdot 10^7$	0.9	12	1.01	1.44
RAI	15	4	$8.36 \cdot 10^7$	$7.27 \cdot 10^6$	$5.65 \cdot 10^6$	$1.50 \cdot 10^8$	2.1	27	1.08	2.25
RAI	21	4	$8.47 \cdot 10^7$	$7.27 \cdot 10^6$	$5.73 \cdot 10^6$	$4.09 \cdot 10^7$	1	6	1.01	1.3
RAI	27	4	$7.14 \cdot 10^7$	$7.27 \cdot 10^6$	$6.49 \cdot 10^6$	$3.56 \cdot 10^7$	1.1	5	1.01	1.22
RAI	32	4	$6.38 \cdot 10^7$	$7.27 \cdot 10^6$	$6.87 \cdot 10^6$	$3.15 \cdot 10^7$	1.1	4	1.01	1.17
GRC	15	3	$2.26 \cdot 10^8$	$2.93 \cdot 10^7$	$2.78 \cdot 10^7$	$1.36 \cdot 10^9$	2.9	52	1.78	4.74
GRC	21	4	$1.42 \cdot 10^8$	$2.93 \cdot 10^7$	$2.60 \cdot 10^7$	$1.65 \cdot 10^8$	1.1	6	1.01	1.23
GRC	27	4	$1.07 \cdot 10^8$	$2.93 \cdot 10^7$	$2.82 \cdot 10^7$	$1.91 \cdot 10^8$	1.1	6	1.01	1.25
GRC	32	4	$9.84 \cdot 10^7$	$2.93 \cdot 10^7$	$2.92 \cdot 10^7$	$1.85 \cdot 10^8$	1.1	6	1.01	1.22
SRR	15	4	$1.17 \cdot 10^8$	$7.90 \cdot 10^7$	$4.93 \cdot 10^7$	$1.34 \cdot 10^{10}$	3.3	96	2.2	20.68
SRR	21	3	$2.39 \cdot 10^9$	$7.35 \cdot 10^7$	$7.09 \cdot 10^7$	$5.63 \cdot 10^8$	1.8	9	1.11	1.68
SRR	27	4	$1.78 \cdot 10^9$	$6.80 \cdot 10^7$	$4.92 \cdot 10^7$	$4.58 \cdot 10^8$	1.3	8	1.04	1.53
SRR	32	4	$1.72 \cdot 10^9$	$6.34 \cdot 10^7$	$4.95 \cdot 10^7$	$4.01 \cdot 10^8$	1.3	7	1.03	1.48

4.2 Set-Min vs Max-Min sketch

We compared Set-Min to Max-Min – an optimized version of Count-Min (see Sect. 2.2). Note that in the case when the k-mer spectrum is strictly decreasing for increasing k-mer counts, the maximum count corresponds to the one with the smallest support. In the general case, we use a variant when, instead of directly comparing count values, counts are ordered according to the support

size. That is, updates are performed by keeping the label with the minimum number of k-mers in the k-mer spectrum, and a query returns the label with the smallest such number.

Table 3 compares Set-Min with Max-Min. As expected from theoretical considerations, the performance of Max-Min in terms of the average error and the sum of errors is better than for regular Count-Min, but worse than for Set-Min. The same behaviour is observed for the number of k-mers having an erroneously estimated frequency. Therefore, Max-Min falls in-between Set-Min and Count-Min, providing a simple and inexpensive practical method to enhance the latter, without reaching the accuracy of the former.

Altogether, Table 3 shows that the performance of Max-Min is closer to Count-Min than to Set-Min. This is because, by keeping the maximum element in each bucket, we are reducing each set of Set-Min to a single element opening the possibility of increased collisions by potentially sharing a given maximum element between unrelated k-mers. The intersection operation performed by Set-Min during query is thus strictly necessary, to guarantee the desired error bounds.

4.3 Set-Min sketch vs KMC output

Not surprisingly, Set-Min achieves better memory consumptions than KMC in all our tests (columns M_{kmc} and M_s of Table 4). Values of R and B do not change from Table 2. The compression rate is variable: from a small factor to two orders of magnitude. The best compression is achieved for larger values of k and assembled genomes. The former is primarily explained by the decreasing number of distinct counts, due to the power-law behaviour. As for the difference between assembled genomes and sequencing data, we will discuss it in more details in Section 5.

Table 3: Set-Min compared to Max-Min sketch. Columns E_c, N_c, A_c are replaced by E_m, N_m, A_m with the same meaning as their Table 2 counterparts.

Name	k	T	E_s	E_m	N_s	N_m	A_s	A_m
SAI	11	$5.50 \cdot 10^4$	$5.00 \cdot 10^4$	$4.15 \cdot 10^5$	1.8	13.5	1.15	1.29
SAI	15	$5.50 \cdot 10^4$	$4.66 \cdot 10^4$	$2.13 \cdot 10^5$	0.9	4	1.01	1.01
SAI	21	$5.50 \cdot 10^4$	$5.29 \cdot 10^4$	$4.02 \cdot 10^5$	0.9	7.2	1.05	1.06
USAI	11	$2.57 \cdot 10^5$	$1.99 \cdot 10^5$	$1.24 \cdot 10^7$	2.6	63.5	2.46	6.4
USAI	15	$2.49 \cdot 10^5$	$2.45 \cdot 10^5$	$1.72 \cdot 10^6$	1.9	13.3	1.31	1.33
USAI	21	$2.38 \cdot 10^5$	$2.00 \cdot 10^5$	$1.76 \cdot 10^6$	1.6	14.2	1.21	1.24
MNO	15	$1.43 \cdot 10^6$	$1.39 \cdot 10^6$	$5.74 \cdot 10^6$	1.4	5.6	1	1.02
MNO	21	$1.43 \cdot 10^6$	$1.41 \cdot 10^6$	$1.78 \cdot 10^7$	1.1	11.2	1.03	1.31
MNO	27	$1.43 \cdot 10^6$	$1.11 \cdot 10^6$	$1.78 \cdot 10^7$	0.9	12.1	1.01	1.2
MNO	32	$1.42 \cdot 10^6$	$1.11 \cdot 10^6$	$1.65 \cdot 10^7$	0.9	11.3	1.01	1.17
RAI	15	$7.27 \cdot 10^6$	$5.65 \cdot 10^6$	$5.81 \cdot 10^7$	2.1	16.8	1.08	1.38
RAI	21	$7.27 \cdot 10^6$	$5.73 \cdot 10^6$	$3.05 \cdot 10^7$	1	5.2	1.01	1.07
RAI	27	$7.27 \cdot 10^6$	$6.49 \cdot 10^6$	$2.84 \cdot 10^7$	1.1	4.5	1.01	1.05
RAI	32	$7.27 \cdot 10^6$	$6.87 \cdot 10^6$	$2.61 \cdot 10^7$	1.1	4	1.01	1.03
GRC	15	$2.93 \cdot 10^7$	$2.78 \cdot 10^7$	$4.07 \cdot 10^8$	2.9	27.8	1.78	2.67
GRC	21	$2.93 \cdot 10^7$	$2.60 \cdot 10^7$	$1.38 \cdot 10^8$	1.1	5.5	1.01	1.08
GRC	27	$2.93 \cdot 10^7$	$2.82 \cdot 10^7$	$1.62 \cdot 10^8$	1.1	6	1.01	1.09
GRC	32	$2.93 \cdot 10^7$	$2.92 \cdot 10^7$	$1.58 \cdot 10^8$	1.1	5.7	1.01	1.08
SRR	15	$7.90 \cdot 10^7$	$4.93 \cdot 10^7$	$3.31 \cdot 10^9$	3.3	61.8	2.2	7.92
SRR	21	$7.35 \cdot 10^7$	$7.09 \cdot 10^7$	$2.42 \cdot 10^8$	1.8	5.9	1.11	1.13
SRR	27	$6.80 \cdot 10^7$	$4.92 \cdot 10^7$	$2.05 \cdot 10^8$	1.3	5.2	1.04	1.06
SRR	32	$6.34 \cdot 10^7$	$4.95 \cdot 10^7$	$1.89 \cdot 10^8$	1.3	4.9	1.03	1.04

4.4 Set-Min sketch vs MPHFs

Table 4 also reports the space usage for `BBHash` to obtain the best memory-optimized hash functions. Column M_{bbhash} is the space (in bytes) required by the hash function only, while M_{bball} is the space required by the hash function plus the external array of frequencies.

As in the previous case, Set-Min sketch is more memory-efficient when k is large, taking about an order of magnitude less memory than a MPHF. For small values of k, `BBHash` takes slightly less space and, being exact, may therefore be

Table 4: Set-Min ($\epsilon = 0.01$) compared to KMC and BBHash (run with $\gamma = 1$). All memory is reported in bytes. Column M_{kmc}, M_s, M_{bball} are the memory taken by a fully functional map between k-mers and their frequencies when applying KMC, Set-Min sketch and `BBHash`, respectively. M_{bbhash} is the memory of the hash function produced by `BBHash` without the external array of frequencies.

Name	k	M_{kmc}	M_s	M_{bbhash}	M_{bball}
SAI	11	$1.21 \cdot 10^7$	$5.75 \cdot 10^6$	$9.13 \cdot 10^5$	$3.00 \cdot 10^6$
SAI	15	$3.80 \cdot 10^7$	$1.20 \cdot 10^6$	$2.06 \cdot 10^6$	$5.99 \cdot 10^6$
SAI	21	$4.77 \cdot 10^7$	$6.77 \cdot 10^5$	$2.03 \cdot 10^6$	$6.00 \cdot 10^6$
USAI	11	$1.54 \cdot 10^7$	$1.49 \cdot 10^7$	$1.17 \cdot 10^6$	$4.61 \cdot 10^6$
USAI	15	$6.95 \cdot 10^7$	$2.87 \cdot 10^7$	$3.72 \cdot 10^6$	$1.35 \cdot 10^7$
USAI	21	$8.53 \cdot 10^7$	$3.00 \cdot 10^7$	$3.91 \cdot 10^6$	$1.39 \cdot 10^7$
MNO	15	$7.08 \cdot 10^8$	$1.68 \cdot 10^8$	$3.87 \cdot 10^7$	$2.15 \cdot 10^8$
MNO	21	$9.80 \cdot 10^8$	$3.55 \cdot 10^7$	$4.66 \cdot 10^7$	$2.45 \cdot 10^8$
MNO	27	$1.24 \cdot 10^9$	$3.75 \cdot 10^7$	$4.73 \cdot 10^7$	$2.48 \cdot 10^8$
MNO	32	$1.37 \cdot 10^9$	$3.84 \cdot 10^7$	$4.90 \cdot 10^7$	$2.36 \cdot 10^8$
RAI	15	$1.76 \cdot 10^9$	$7.54 \cdot 10^8$	$9.59 \cdot 10^7$	$5.98 \cdot 10^8$
RAI	21	$4.37 \cdot 10^9$	$6.78 \cdot 10^8$	$2.16 \cdot 10^8$	$1.24 \cdot 10^9$
RAI	27	$6.04 \cdot 10^9$	$5.36 \cdot 10^8$	$2.37 \cdot 10^8$	$1.29 \cdot 10^9$
RAI	32	$6.96 \cdot 10^9$	$4.79 \cdot 10^8$	$2.70 \cdot 10^8$	$1.38 \cdot 10^9$
GRC	15	$3.83 \cdot 10^9$	$1.70 \cdot 10^9$	$2.09 \cdot 10^8$	$1.65 \cdot 10^9$
GRC	21	$1.86 \cdot 10^{10}$	$1.28 \cdot 10^9$	$9.32 \cdot 10^8$	$6.46 \cdot 10^9$
GRC	27	$2.48 \cdot 10^{10}$	$9.66 \cdot 10^8$	$9.86 \cdot 10^8$	$6.57 \cdot 10^9$
GRC	32	$2.82 \cdot 10^{10}$	$8.38 \cdot 10^8$	$1.08 \cdot 10^9$	$6.54 \cdot 10^9$
SRR	15	$4.73 \cdot 10^9$	$2.00 \cdot 10^9$	$2.59 \cdot 10^8$	$2.12 \cdot 10^9$
SRR	21	$2.91 \cdot 10^{10}$	$1.61 \cdot 10^{10}$	$1.48 \cdot 10^9$	$1.10 \cdot 10^{10}$
SRR	27	$3.73 \cdot 10^{10}$	$1.60 \cdot 10^{10}$	$1.48 \cdot 10^9$	$1.08 \cdot 10^{10}$
SRR	32	$4.07 \cdot 10^{10}$	$1.46 \cdot 10^{10}$	$1.57 \cdot 10^9$	$1.04 \cdot 10^{10}$

the preferable choice. However, one should keep in mind that MPHF does not support updates, while a Set-Min sketch is updatable to a certain extent with new (k-mer,count) pairs, and also mergeable with another possibly redundant map.

The behaviour of the unassembled datasets is of particular interest. Even for large k's, MPHF appears to be a better choice for this type of data. The causes

of this phenomenon and possible solutions are discussed in Section 5.

4.5 Time measurements

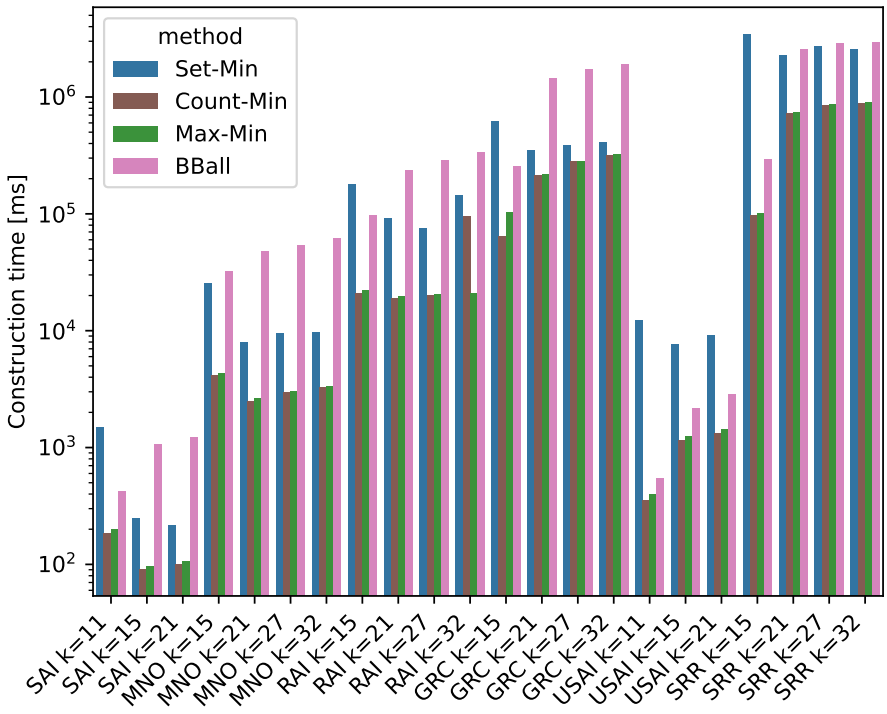

Figure 3: Construction time of Set-Min sketches compared to Count-Min, Max-Min and BBHash (with external array). Time is reported in milliseconds on a logarithm scale.

Construction time is reported in Figure 3. Set-Min sketches are generally faster to build than memory-optimized BBHash except for smaller values of k. However, similarly to Table 4, Set-Min sketch is at a disadvantage when the count value distribution is less skewed, such in the case of short k or for unassembled reads. For highly skewed data, Set-Min can be built faster than BBHash MPHFs, but is still more computationally demanding than Count-Min or Max-Min because of the additinal operations required to create and update

the sets of labels.

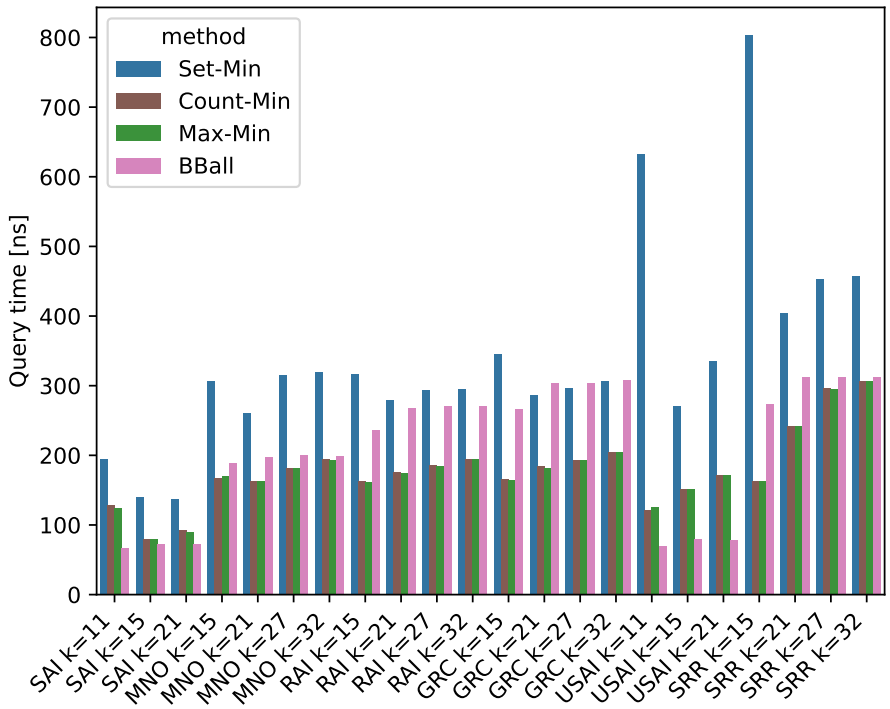

Figure 4: Average query time of Set-Min, Count-Min and BBHash.

Set-Min average query time performance is 50% slower than Count-Min (and by extention of Max-Min) and comparable to those of BBHash, when data is very skewed. Following the previous trend, Set-Min sketches appear to be the slowest method for small values of k and for unassembled reads.

5 Discussion

5.1 Unassembled datasets

As seen in Table 4, for the unassembled datasets, Set-Min sketch does not seem to have an advantage in memory usage, even for large k's. We found

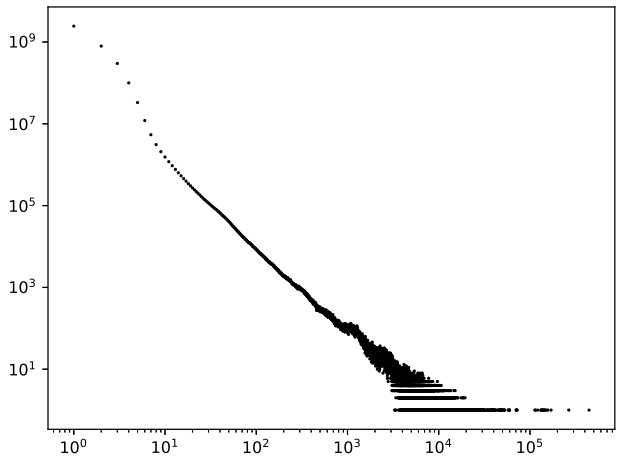

Figure 5: Spectrum in log-log scale of SRR unassembled data sets for $k = 32$.

that this is due to low-count k-mers, specifically to k-mers whose count does not exceed the sequencing coverage. It is known that for Illumina sequencing, sequencing errors produce a linear growth of the number of new distinct k-mers (for large k) depending on the coverage (see e.g. Figure 2(b) of (Salikhov *et al.*, 2014)). Frequencies of these "erroneous" k-mers do not have the same statistical behaviour as *bona fide* k-mers, in particular first spectrum values do not decay at the same rate as the rest of the spectrum. Figure 5 shows the spectrum of the unassembled SRR datasets. One can observe a slower decay behaviour for a few first spectrum values. In this situation, additional rows are needed just to make the sketch able to distinguish, with required precision, between small frequency values. Note that in practice, distinguishing between small frequencies is often irrelevant. For example, many read assemblers simply discard low-frequency reads as a way to de-noise the data. In the case of Set-Min, it is possible to collapse together the first m columns of the spectrum by assigning to all k-mers in this subset the same frequency. This would considerably reduce the sketch

size. Formally, error guarantee (5) would not hold anymore, but most of newly introduced errors would be small (typically, equal to 1) and would occur for low counts only.

To check the above, we constructed a Set-Min sketch for SRR with dimensions $(R, B) = (4, 3310557)$, merging together the first five columns of the sorted spectrum and assigning count 5 to all merged k-mers. While the final sum of errors was well above the theoretical limit (10^9 against $7 \cdot 10^6$), the maximum and average error were respectively 55 and 2.8. In many applications this error level could be acceptable.

5.2 Presence-absence information

As introduced in Section 3.2, in this work we assumed that only k-mers present in the dataset can be queried. This assumption allowed us to discard the largest value of the spectrum corresponding to unique k-mers, thereby saving space. Set-Min sketches can seamlessly work without this assumption, but the space required for storing k-mer counters may not be competitive to other solutions. An alternative could be to build an additional data structure, such as a Bloom filter, representing presence-absence information for the set of k-mers having the largest count. This allows the discrimination between k-mers absent in the dataset from those present but non-represented in the sketch.

Another scenario occurs when working with multiple datasets of very high similarity, such as a large collection of bacterial strains or a collection of RNA-seq data (Solomon and Kingsford, 2016; Yu *et al.*, 2018). In this case, it might be beneficial to build a Bloom filter for the k-mers present in the union of the datasets, and maintain multiple Set-Min sketches to represent k-mer counts in each dataset.

Note that Set-Min sketches can also be helpful for long-term storage and

transmission of the k-mer composition of a dataset augmented with count information. The k-mers of the dataset can be reassembled into simplitigs (Břinda *et al.*, 2020) with a Set-Min sketch storing the (approximated) frequencies. The full count table can be restored from the simplitigs and the sketch.

6 Conclusions

We presented Set-Min sketch – a novel sketching method inspired by the Count-Min sketch. Its primary use is to associate keys to labels without explicitly storing the former. In this paper, we demonstrated the performance of Set-Min sketch for storing k-mer counts information, where the distribution of labels (k-mer counts) follows a power-law distribution. Under this assumption, we proposed simple bounds for a Set-Min sketch that guarantee the total error sum to be within an ε fraction of the total number of k-mers in the dataset.

We showed that Set-Min sketch allows us to save space compared to the raw output of the popular KMC k-mer counting tool when applied to labels following a skewed distribution, at the price of a very modest error rate. This saving is especially important in the case of whole-genome data and large values of k, where it can achieve a two orders of magnitude reduction in memory usage. Set-Min has been shown to be more space efficient than the MPHF-based soluton for large values of k. For smaller k's, however, MPHFs provide an implementation with comparable memory consumption. Finally, when compared to Count-Min and Max-Min sketches of comparable dimensions, our sketch achieves better point-query errors thanks to the distribution-aware dimensioning performed on the k-mer spectrum, and the reliance on already computed count tables as input.

Acknowledgement

The authors are grateful to the RECOMB anonymous reviewers for their helpful comments.

Author Disclosure Statement

The authors declare they have no conflicting financial interests.

Funding Information

GK was partially funded by RFBR, project 20-07-00652, and joint RFBR and JSPS project 20-51-50007.

References

Adamic, L.A. 2000. Zipf, Power-laws, and Pareto - a ranking tutorial.

Benoit, G., Peterlongo, P., Mariadassou, M., *et al.* 2016. Multiple comparative metagenomics using multiset k-mer counting. *PeerJ Computer Science* 2, e94.

Břinda, K., Baym, M., and Kucherov, G. 2020. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *bioRxiv* , 2020.01.12.903443.

Chor, B., Horn, D., Goldman, N., *et al.* 2009. Genomic dna k-mer spectra: models and modalities. *Genome Biology* 10, 10, R108.

Chum, O., Philbin, J., and Zisserman, A. 2008. Near Duplicate Image Detection: min-Hash and tf-idf Weighting. In *Procedings of the British Machine Vision Conference 2008*. British Machine Vision Association, Leeds. pages 50.1–50.10.

Cormode, G. 2009. Count-Min Sketch. In L. Liu and M.T. Özsu, editors, *Encyclopedia of Database Systems*. Springer US, Boston, MA. pages 511–516.

Cormode, G. and Muthukrishnan, S. 2005a. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1, 58–75.

Cormode, G. and Muthukrishnan, S. 2005b. Summarizing and Mining Skewed Data Streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics. pages 44–55. 5th SIAM International Conference on Data Mining, SDM 2005 ; Conference date: 21-04-2005 Through 23-04-2005.

Csűrös, M., Noé, L., and Kucherov, G. 2007. Reconsidering the significance of genomic word frequencies. *Trends in Genetics* 23, 11, 543–546.

Esposito, E., Graf, T.M., and Vigna, S. 2019. RecSplit: Minimal Perfect Hashing via Recursive Splitting. *arXiv:1910.06416 [cs]* ArXiv: 1910.06416.

Estan, C. and Varghese, G. 2002. New directions in traffic measurement and accounting. In M. Mathis, P. Steenkiste, H. Balakrishnan, and V. Paxson, editors, *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 19-23, 2002, Pittsburgh, PA, USA*. ACM. pages 323–336.

Hatje, K. and Kollmar, M. 2012. A Phylogenetic Analysis of the Brassicales Clade Based on an Alignment-Free Sequence Comparison Method. *Frontiers in Plant Science* 3.

Holley, G. and Melsted, P. 2019. Bifrost – Highly parallel construction and indexing of colored and compacted de Bruijn graphs. *bioRxiv* , 695338.

Khorsand, P. and Hormozdiari, F. 2019. Nebula: Ultra-efficient mapping-free structural variant genotyper. *bioRxiv* , 566620.

Kokot, M., Długosz, M., and Deorowicz, S. 2017. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics* 33, 17, 2759–2761.

Limasset, A., Rizk, G., Chikhi, R., *et al.* 2017. Fast and scalable minimal perfect hashing for massive key sets. *arXiv:1702.03154 [cs]* ArXiv: 1702.03154.

Liu, Y., Schröder, J., and Schmidt, B. 2012. Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics* 29, 3, 308–315.

Marçais, G. and Kingsford, C. 2011. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics* 27, 6, 764.

Müller, I., Sanders, P., Schulze, R., *et al.* 2014. Retrieval and Perfect Hashing Using Fingerprinting. In J. Gudmundsson and J. Katajainen, editors, *Experimental Algorithms*. Lecture Notes in Computer Science. Springer International Publishing, Cham. pages 138–149.

Rahman, A. and Medvedev, P. 2020. Representation of k-mer sets using spectrum-preserving string sets. *bioRxiv* , 2020.01.07.896928.

Rahman, A., Hallgrímsdóttir, I., Eisen, M., *et al.* 2018. Association mapping from sequencing reads using k-mers. *eLife* 7, e32920.

Rizk, G., Lavenier, D., and Chikhi, R. 2013. DSK: k-mer counting with very low memory usage.

Salikhov, K., Sacomoto, G., and Kucherov, G. 2014. Using cascading Bloom filters to improve the memory usage for de Brujin graphs. *BMC Algorithms for Molecular Biology* 9, 1, 2.

Solomon, B. and Kingsford, C. 2016. Fast Search of Thousands of Short-Read Sequencing Experiments. *Nature biotechnology* 34, 3, 300.

Yi, H. and Jin, L. 2013. Co-phylog: an assembly-free phylogenomic approach for closely related organisms. *Nucleic Acids Research* 41, 7, e75.

Yu, Y., Belazzougui, D., Qian, C., *et al.* 2017. Memory-efficient and Ultra-fast Network Lookup and Forwarding using Othello Hashing. *arXiv:1608.05699 [cs]* ArXiv: 1608.05699.

Yu, Y., Liu, J., Liu, X., *et al.* 2018. SeqOthello: querying RNA-seq experiments at scale. *Genome Biology* 19, 1, 167.

Zielezinski, A., Girgis, H.Z., Bernard, G., *et al.* 2019. Benchmarking of alignment-free sequence comparison methods. *Genome Biology* 20, 1, 144.