Enhanced Compression of k-mer Sets with Counters via de Bruijn Graphs

Enrico Rossignolo, Matteo Comin*
Department of Information Engineering, University of Padua, Italy,
{enrico.rossignolo,matteo.comin}@unipd.it

January 25, 2025

Keywords: k-mer set with counts; compression; de Bruijn Graph; smallest path cover

Abstract: An essential task in computational genomics involves transforming input sequences into their constituent k-mers. The quest for an efficient representation of k-mer sets is crucial for enhancing the scalability of bioinformatics analyses. One widely used method involves converting the k-mer set into a de Bruijn graph, followed by seeking a compact graph representation via the smallest path cover.

This paper introduces USTAR, a tool designed to compress both a set of k-mers and their associated counts. USTAR leverages the connectivity and density of de Bruijn graphs, enabling a more efficient path selection for constructing the path cover. The efficacy of USTAR is demonstrated through its application in compress-

*Corresponding author: Matteo Comin `comin@dei.unipd.it`

ing real read datasets. USTAR improves the compression achieved by UST, the best algorithm, by percentages ranging from 2.3% to 26.4%, depending on the k-mer size, and it is up to 7× times faster. The USTAR code is available at the repository `https://github.com/CominLab/USTAR`.

1 Introduction

Most bioinformatics analyses heavily rely on k-mer-based tools, offering numerous advantages over methods directly processing individual reads or read alignments. These tools operate primarily by converting the input sequence data, whose lengths vary depending on the sequencing technology utilized, into a fixed-length set of strings known as k-mers. In many applications, the k-mer set includes also the counts (i.e. multiplicities) associated with each k-mer. Methods based on k-mers exhibit superior performance across various applications. For instance, in genome assembly, Spades (Bankevich et al., 2012) utilizes k-mers-based techniques to accurately reconstruct entire genomes from reads, achieving highly efficient and precise results. Additionally, Merqury (Rhie et al., 2020) employs k-mer counts for assembly validation. In the area of metagenomics, Kraken (Wood and Salzberg, 2014) effectively classifies and identifies microorganisms within complex environmental samples by leveraging k-mers, and it is 900 times faster than MegaBLAST. Consequently, the advent of Kraken has spurred the development of numerous metagenomic classification tools predominantly based on k-mers (Andreace et al., 2021; Qian and Comin, 2019; Cavattoni and Comin, 2023; Storato and Comin, 2022). Moreover, in genotyping, various tools (Denti et al., 2019; Sun and Medvedev, 2019; Marcolin et al., 2022; Monsu and Comin, 2021) use k-mers instead of alignment techniques to identify genetic variations in individuals or populations. Furthermore, in phy-

logenomics, Mash (Ondov et al., 2016) efficiently estimates distances between genomes and metagenomes by utilizing k-mers, in order to reconstruct the evolutionary relationships among organisms. Numerous methods based on k-mers have been proposed in database searching to enhance sequence search efficiency (Sun et al., 2018; Harris and Medvedev, 2020; Bradley et al., 2019; Pandey et al., 2018a; Marchet et al., 2020).

K-mer-based methodologies have fundamentally transformed various areas of bioinformatics, emerging as indispensable tools for analyzing extensive genomic data. These tools often rely on specialized data structures to represent sets of k-mers (for an overview, refer to (Chikhi et al., 2021)). However, accommodating these data structures within modern sequencing datasets, which are huge, presents a bottleneck in scaling up for larger databases. Conway and Bromage (Conway and Bromage, 2011) have demonstrated that at least $\log \binom{4^k}{n}$ bits are necessary to store a set of n k-mers in the worst-case scenario. Yet, sets of k-mers derived from sequencing experiments typically exhibit a spectrum-like property (Chikhi et al., 2021), often containing redundant information. Consequently, in practical terms, most data structures can notably surpass this lower bound (Chikhi et al., 2016b). Given that the storage of a k-mer set demands significant space, it is preferable to minimize its size, especially when it can be considerably large. For instance, the dataset employed to evaluate the BIGSI (Rahman and Medvedev, 2020) index necessitates approximately 12 TB for storage in a compressed format.

The most effective tool for compressing a set of k-mers with counts is UST (Rahman and Medvedev, 2020) (see Section 1.1), leveraging the de Bruijn graph representation of the input k-mer set. The challenge of determining the most compact representation of a k-mer set is analogous to identifying the smallest path cover within a de Bruijn graph (as detailed in Section 2).

In this paper, we introduce USTAR (Unitig STitch Advanced constRuction) [1], which follows a similar paradigm but integrates a better strategy to navigate de Bruijn graphs. The USTAR algorithm leverages the density of the de Bruijn graph and node connectivity, facilitating a more efficient path selection for constructing the path cover and consequently enhancing compression.

In Section 3 we reported a series of results on several real sequencing datasets. The results demonstrate that USTAR achieves the best compression ratio for both k-mers and counts, surpassing the performance of UST and other existing tools.

1.1 Related works

The problem of k-mer set compression has been addressed by several researchers, in this section we provide a summary of the most recent findings.

K-mer counters serve as tools explicitly engineered to count and store distinct k-mers, a particularly hard challenge for large datasets. Among the most famous tools in this domain are Squeakr (Pandey et al., 2018b), KMC (Kokot et al., 2017), and DSK (Rizk et al., 2013). Squeakr operates as both an approximate and exact k-mers counting system, utilizing Bloom filters—an efficient probabilistic data structure—to store k-mers effectively. KMC partitions, sorts, and counts k-mers by employing disk files as bins. Meanwhile, DSK manages k-mer counters using hash tables for updates. Although the latter two tools are not explicitly designed for compression, they are capable of reducing size by storing only unique k-mers and their respective counts.

A k-mer set with counters can be represented using a de Bruijn graph (dBG), which can be effectively utilized for space-efficient storage. BCALM2 (Chikhi et al., 2016a) serves as a tool designed for constructing memory-efficient dBGs

[1] A preliminary version of USTAR has been presented at ISBRA 2023 (Rossignolo and Comin, 2023).

that undergo compaction, a process where maximal non-branching paths are combined into a single node labeled with concatenated k-mers and their associated count list. Compaction not only offers benefits in terms of reduced memory usage but also in terms of disk space. The concept behind compressing k-mers involves saving $k-1$ characters per link. For instance, within a dBG, a non-branching path like (ACT, CTG, TGA) could be replaced by a unified node labeled as $ACTGA$. This sequence, represented by a non-branching path, is called unitig, and it is an attempt to compress k-mers using dBGs.

Another method to minimize the redundancy within a k-mer set involves leveraging its spectrum-like property (Chikhi et al., 2021), i.e. the existence of long strings that "generate" all the k-mers. This concept has concurrently been explored by the authors of ProphAsm (Břinda et al., 2021) and UST (Rahman and Medvedev, 2020). In their work, the authors of ProphAsm (Břinda et al., 2021) refer to these long strings as simplitigs. They construct these simplitigs by merging overlapping unitigs and k-mers while dynamically exploring a de Bruijn graph (dBG). Their findings demonstrated that simplitigs surpassed unitigs, the k-mer representation proposed by BCALM2, in terms of computational efficiency and compression rates.

Similarly, UST (Rahman and Medvedev, 2020) links overlapping unitigs and k-mers. However, it utilizes the compacted de Bruijn graph generated by BCALM2 as input, also taking into consideration k-mer counts. They found a nearly tight lower bound for the best k-mer representation and empirically showcased that, in most scenarios, their greedy algorithm remains within a 3% margin of this lower bound.

Both sets of authors have established a lower bound on the cumulative length of simplitigs and demonstrated that their heuristics produce representations with a cumulative length closely approaching the lower bound for typical k-

values, e.g. 31. When k-values are lower (e.g., < 20), resulting in denser de Bruijn graphs, their heuristic does not approach the lower bound as closely as for larger k-values.

All these authors have explored whether computing minimum simplitigs might be NP-hard. This has been recently disproved by Schmidt and Alanko (2022), who have shown that simplitigs with minimum cumulative length, named also Eulertigs, can be computed in polynomial time. However, Eulertigs are only slightly smaller than strings computed by previous heuristics, and the resulting compression is not guaranteed to be optimal.

2 USTAR: Unitig STitch Advanced constRuction

2.1 Definitions

In this paper, we focus on strings composed of the characters A, C, T, and G. A string of length k is called a k-mer. The reverse complement of a k-mer, denoted by $rc(k-mer)$, is obtained by reversing the k-mer and replacing each character with its complementary character: A becomes T, C becomes G, T becomes A, and G becomes C. Since we don't have access to the specific DNA strand from which the k-mer is derived, we consider a k-mer and its reverse complement as equivalent.

Given a string $s =< s_1, ..., s_{|s|} >$, the first i characters are represented by $pref_i(s) =< s_1, ..., s_i >$, and the last i characters are denoted by $suf_i(s) =< s_{|s|-i+1}, ..., s_{|s|} >$.

We define the operation of gluing two strings u and v, where $suf_{k-1}(u) = pref_{k-1}(v)$, as the concatenation of u with the suffix of v:

$$u \odot^{k-1} v = u \cdot suf_{|v|-(k-1)}(v)$$

For instance, given two 3-mers $u = CTG$ and $v = TGA$, their gluing is $u \odot^2 v = CTGA$.

A collection of k-mers can be effectively represented using a de Bruijn graph, which we will define in a node-centric manner, where the connectivity of the graph is implicitly determined by the nodes themselves. This approach allows for a seamless interchange between referring to k-mers and the associated de Bruijn graph representation $dBG(K)$.

For a given k-mer set $K = \{m_1, ..., m_{|K|}\}$, the corresponding de Bruijn graph $dBG(K) = (V, A)$ exhibits the following characteristics:

1. Vertex Set : V represents the set of all k-mers in K.

2. Arc Set : A defines the set of directed arcs connecting the k-mers in K.

Arcs are established based on the k-mer overlap principle: an arc (u, v) exists in A if and only if the suffix of u coincides with the prefix of v. This ensures that the de Bruijn graph accurately captures the connectivity patterns within the k-mer set.

More formally, given a k-mer set $K = \{m_1, \dots m_{|K|}\}$, a de Bruijn graph of K is a directed graph dBG(K)=(V, A) in which:

1. $V = \{v_1, \dots, v_{|K|}\}$

2. each node $v \in V$ has a label $lab(v_i) = m_i$

3. each node $v \in V$ has two different sides $s_v \in \{0, 1\}$, where $(v, 1)$ is graphically represented with a tip

4. a node side (v, s_v) is spelled as

$$spell(v, s_v) = \begin{cases} lab(v) & s_v = 0 \\ rc(lab(v)) & s_v = 1 \end{cases} \tag{1}$$

5. there is an arc between two node sides (v, s_v) and (u, s_u) if and only if there are spellings that share a $(k-1)$-mer. In particular, it must be

$$\big((v, s_v), (u, s_u)\big) \in A \iff suf_{k-1}\left(spell\left(v, 1 - s_v\right)\right) = pref_{k-1}\left(spell\left(u, s_u\right)\right)$$

The right-hand condition is also known as (v, u)-oriented-overlap (Rahman and Medvedev, 2020).

De Bruijn graphs represent k-mers and their reverse complements using corresponding nodes, allowing for efficient handling of both forward and reverse directions of a sequence. Additionally, nodes can be associated with k-mer counts, providing information about the frequency of occurrence of each k-mer in the input data.

The spelling of a path $p =< (v_1, s_1), ..., (v_l, s_l) >$ is determined by concatenating the spellings of its constituent nodes, adhering to the k-mer overlap principle:

$$spell(p) = spell(v_1, s_1) \odot^{k-1} spell(v_2, s_2) \odot^{k-1} \cdots \odot^{k-1} spell(v_l, s_l)$$

A unitig in the de Bruijn graph is a path $p =< (v_1, s_1), ..., (v_l, s_l) >$ whose internal nodes have in-degree and out-degree equal to 1. This implies that each node is directly connected to only one other node in the path. A maximal unitig cannot be extended further in either direction.

To optimize memory usage, de Bruijn graphs can be compacted by replacing

maximal unitigs with single nodes labeled with the spellings of the unitigs. This compressed representation preserves the essential connectivity information while reducing graph size.

Figure 1 illustrates an example of a compacted de Bruijn graph corresponding to the k-mer set $K = \{ACT, CTG, TGA, CTT, TTG, TGC\}$. The maximal unitig (CTT, TTG) has been replaced with the node labeled $CTTG$.

2.2 Vertex-Disjoint Path Cover Problem

Compressing a k-mer set K entails constructing a representation S of K using strings of varying lengths, ensuring that the set of its substrings of length k comprises all the k-mers in K.

A common method for assessing the size of a string set S is by calculating its *cumulative length*, which represents the sum of all the string lengths.

In Rahman and Medvedev (2020) and Břinda et al. (2021) it has been shown that, when a representation S of a k-mer set K doesn't include duplicate k-mers, its cumulative length is directly proportional to its cardinality. This relationship can be expressed as follows:

$$CL(S) = |K| + (k-1) \cdot |S|$$

where $|S|$ represents the cardinality of the set S. Consequently, our goal of identifying the most efficient representation for a k-mer set K can be reframed as minimizing the number of strings in the string set S.

Let's revisit the example in Figure 1. From the path $p = (ACT, CTG, TGA)$, we can derive its spell $spell(p) = ACTGA$, which includes all the k-mers ACT, CTG, and TGA present in p. By considering a set of paths P that encompasses all the nodes in dBG(K), we can construct a set S of compressed k-mers. By enforcing the property that all paths within P are vertex-disjoint, we ensure

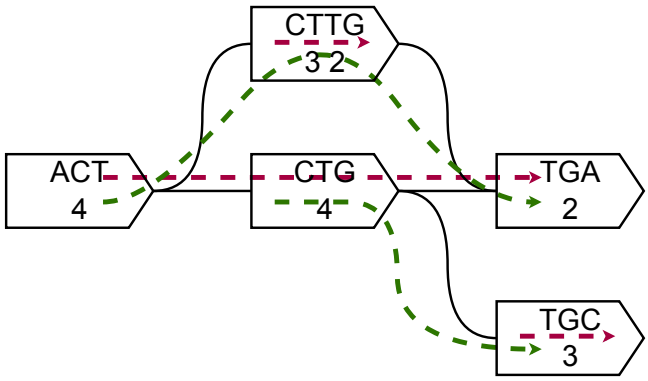


Figure 1: An example of a compacted de Bruijn graph. Each node is annotated with k-mers and their respective counts. Instead of employing two arcs with opposing directions, undirected arcs are utilized. In this example, the UST algorithm opts for the path cover highlighted in red, whereas USTAR is will select the path cover highlighted in green.

that each k-mer appears only once in S. As a result, a vertex-disjoint path cover can be effectively utilized to determine the set S for compression.

To achieve a compact representation of a k-mer set K, we aim to minimize the cumulative length of a string set S that represents K. This is equivalent to minimizing the number of strings $|S|$, which in turn corresponds to the number of vertex-disjoint paths $|P|$ in a de Bruijn graph dBG(K) associated with K. This problem is formally defined as the *minimum vertex-disjoint path cover problem* for dBG(K).

Problem 1. *Given a de Bruijn graph dBG(K) of a k-mer set K, the* ***minimum vertex-disjoint path cover problem*** *is to find the minimum number of vertex-disjoint paths that cover the graph.*

The minimum vertex-disjoint path cover problem is known to be NP-hard for general graphs (Břinda et al., 2021; Rahman and Medvedev, 2020), as it can be reduced from the Hamiltonian path problem. However, for de Bruijn graphs, efficient algorithms exist based on the concept of Eulerian paths (Schmidt and

Alanko, 2022), named also Euleritigs. However the Euleritigs are not suited for incorporating count values.

Although optimal algorithms are available, simpler greedy approaches have also been proposed. ProphAsm (Břinda et al., 2021) uses a simple heuristic that takes an arbitrary k-mer in the dBG(K), and it tries to extend it forward and backward as long as possible and it restarts until it consumes all the k-mers. Similarly, using as input the compacted dBG(K) constructed by BCALM2, UST (Rahman and Medvedev, 2020) takes an arbitrary node, tries to extend it forward as long as possible, and restarts until there are available nodes. In the end, UST merges linked paths. Both methods perform a similar strategy by picking the first available k-mer, and without considering the graph structure.

Consider the example in Figure 1. ProphAsm and UST would construct a path cover (in red) $P = \{(ACT, CTG, TGA), (CTT, TTG), (TGC)\}$, resulting in a string set $S = \{ACTGA, CTTG, TGC\}$ with cumulative length $CL(S) = 12$.

2.3 USTAR Algorithm

The ability to efficiently represent k-mer sets with counts is crucial for genome assembly and other applications in computational biology. In this work, we introduce USTAR (Unitig STitch Advanced constRuction), a novel algorithm that leverages the connectivity and count information of de Bruijn graphs to achieve more compact k-mer set representations. Unlike existing approaches, USTAR employs a strategic seed selection mechanism and a path extension strategy that optimizes the cumulative length (CL) of the resulting string set.

Similar to previous methods, USTAR utilizes the compacted de Bruijn graph derived from BCALM2. At each iteration, USTAR selects a seed node based on its connectivity and count values. USTAR begins by choosing a seed node

within the graph, then attempts to construct a path originating from this node. This path is formed by linking neighboring nodes until it reaches its maximum extension. The algorithm proceeds by selecting a fresh seed node until all nodes have been crossed by a path. The fundamental operations in this algorithm involve determining an optimal seed node selection and effectively elongating a path among the accessible connections.

In USTAR, leveraging both the node-associated counts and the topological properties of the graph aids in identifying a good seed node and in designing strategies for its extension. Notably, the count distribution tends to exhibit significant skewness, characterized by numerous low values and relatively few high values. This non-uniform and skewed distribution indicates that compressing higher counts poses greater difficulty. The selection process prioritizes nodes with higher average counts, as these nodes typically represent longer strings and are crucial for achieving efficient compression. Typically, neighboring nodes exhibit similar counts, suggesting that selecting the seed node based on the highest average count could enhance the compression of these elevated counts.

In the context of constructing path covers, we note that both the UST and ProphAsm algorithms may select highly connected nodes initially. However, since these nodes become unavailable in subsequent iterations, this choice can result in isolated nodes. This, in turn, leads to an increase in the overall cumulative length of the path cover, as illustrated in Figure 1. Instead, in USTAR, the path extension is guided by a strategy that avoids selecting highly connected nodes, ensuring that such nodes remain available for future iterations. This approach minimizes the formation of isolated nodes and reduces the overall CL, since it creates fewer and longer simplitigs.

The pseudocode of USTAR is presented in Algorithm 1. The algorithm begins by sorting the nodes in decreasing order of average count to prioritize

nodes with higher counts. Next, a non-visited node is selected and used as a seed to initiate path generation. The path is extended in both directions, starting from the seed node, by selecting the least connected non-visited adjacent node as the successor. This strategy ensures that highly connected nodes remain available for future path construction, minimizing isolated nodes and lowering the overall cumulative length (CL). As the path extends, all traversed nodes are marked as visited to prevent their re-utilization.

Algorithm 1: USTAR

```
Data: de Bruijn graph dBG
Result: Sequences set S
begin
    S = ∅
    seed-nodes = sort nodes by average counts
    for seed ∈ seed-nodes do
        if seed is not visited then
            visit(seed)
            contig = Extend(seed) to the right
            contig = Extend(contig) to the left
            S = S ∪ {contig}
        end
    end
    return S
end
Function Extend(contig):
    L = {non-visited neighbors of contig head}
    while L not empty do
        v = less connected node in L
        visit(v)
        contig = merge(v, contig)
        L = {non-visited neighbors of v}
    end
    return contig
```

This seed selection and path extension strategy distinguishes USTAR from previous methods and contributes to its superior compression performance. By strategically utilizing the connectivity and count information of de Bruijn graphs, USTAR effectively constructs compact representations of k-mer sets.

Following the example in Figure 1, USTAR guarantees that while constructing the first path, the most connected node CTG is avoided. This will produce a cover of the dBG with the paths (in green) $P' = \{(ACT, CTT, TTG, TGA), (CTG, TGC)\}$ and thus a set of strings $S' = \{ACTTGA, CTGC\}$. If we measure the cumulative length of S' we have that $CL(S') = 10$.

In this particular scenario, the uncompressed set of k-mers K has a cumulative length of $CL(K) = 18$. Using the UST algorithm, the k-mer set can be compressed to the sequence set S achieving $CL(S) = 12$, while, employing USTAR, we achieve even better compression results, reducing the length further to $CL(S') = 10$ for the sequence set S':

$$CL(S') = 10 < CL(S) = 12 < CL(K) = 18$$

3 Results

In this section, we present a series of experiments in order to find the best tool that compresses k-mers and counts. In our evaluation, we compared USTAR with several other tools: Squeaker (Pandey et al., 2018b), KMC (Kokot et al., 2017), DSK (Rizk et al., 2013), BCALM2 (Chikhi et al., 2016b), and UST (Rahman and Medvedev, 2020). We used for testing a set of real reads datasets taken from previous studies (Pandey et al., 2018b; Rizk et al., 2013; Kokot et al., 2017; Chikhi et al., 2016b; Břinda et al., 2021). An overview of all datasets is presented in Table 1. For each dataset, we gathered all k-mers (refer to Appendix Table 6) along with their respective counts, utilizing this data as input for all compression techniques. Certain tools, such as UST and USTAR, need the execution of BCALM2 as a preprocessing step in order to construct the compacted de Bruijn graph.

name	description	read length	#reads	size [GB]
SRR001665	Escherichia coli	36	20,816,448	9.304
SRR061958	Human Microbiome 1	101	53,588,068	3.007
SRR062379	Human Microbiome 2	100	64,491,564	2.348
SRR10260779	Musa balbisiana RNA-Seq	101	44,227,112	2.363
SRR11458718	Soybean RNA-seq	125	83,594,116	3.565
SRR13605073	Broiler chicken DNA	92	14,763,228	0.230
SRR14005143	Foodborne pathogens	211	1,713,786	0.261
SRR332538	Drosophila ananassae	75	18,365,926	0.683
SRR341725	Gut microbiota	90	25,479,128	1.254
SRR5853087	Danio rerio RNA-Seq	101	119,482,078	3.194
SRR957915	Human RNA-seq	101	49,459,840	3.671

Table 1: A summary of the read datasets used in the experiments. Datasets are downloaded from NCBI's Sequence Read Archive.

3.1 Preliminary Experiments: Compression of k-mer Sets Without Counts

Before assessing the quality of compression of k-mer sets with counts, we run a preliminary experiment to test just the compression of k-mer sets. Since we know that the algorithm used by Eulertigs gives the optimal solution in terms of cumulative length, we want to compare it with UST, one of the best compressors, and with USTAR.

In Table 2 are shown the CL obtained with UST, USTAR and Eulertigs when compressing the k-mer sets for all input files. In the Table we also reported the non-tight CL lower bound defined in (Rahman and Medvedev, 2020), and the size of the compressed files using MFCompress (Pinho and Pratas, 2014). As expected, Eulertigs obtains the minimum CL for all datasets. It is interesting to note that these optimal values of CL are in line with the theoretical non-tight lower bound. In four cases USTAR reached the optimum CL and in many cases it is close to the optimum, while UST consistently obtained the worst CL. As for the compression, we can note that even if USTAR did not obtain the best CL, it produces the smallest files on average. The CL can indeed be used as a proxy for

		CL			Compression		
Dataset	**CL_LB**	**UST**	**USTAR**	**Eulertigs**	**UST**	**USTAR**	**Eulertigs**
SRR001665_1	36,266,668	36,357,088	36,324,848	36,324,848	10,010,780	9,851,087	9,954,277
SRR001665_2	45,579,202	45,751,302	45,694,102	45,694,102	12,648,723	12,441,689	12,593,563
SRR061958_1	190,402,386	193,356,086	191,039,546	191,038,846	50,773,252	50,063,473	50,733,020
SRR061958_2	210,715,649	213,767,349	211,459,069	211,458,409	56,260,349	55,561,835	56,331,236
SRR062379_1	246,722,575	252,423,135	248,519,195	248,517,935	68,013,430	66,895,296	67,436,683
SRR062379_2	239,342,674	246,067,794	241,478,854	241,477,514	65,785,633	64,550,948	65,024,603
SRR10260779_1	183,310,348	187,998,708	184,854,128	184,851,568	49,700,527	48,786,536	49,060,336
SRR10260779_2	208,110,183	214,248,923	210,202,863	210,200,303	56,865,823	55,702,332	56,053,793
SRR11458718_1	183,078,881	189,836,581	185,070,621	185,068,101	49,403,003	48,205,642	48,920,148
SRR11458718_2	194,338,234	202,875,974	196,865,774	196,863,334	52,704,066	51,518,947	51,884,673
SRR13605073_1	84,333,700	86,012,760	84,974,600	84,973,100	21,495,245	21,210,888	21,348,205
SRR14005143_1	18,995,319	19,353,239	19,020,479	19,020,479	5,016,614	4,933,684	4,897,819
SRR14005143_2	41,370,913	42,325,793	41,492,693	41,492,693	11,018,069	10,826,943	10,824,148
SRR332538_1	18,243,707	18,649,867	18,382,827	18,382,407	4,697,479	4,668,764	4,671,119
SRR332538_2	45,168,450	49,636,030	46,689,430	46,689,210	12,102,067	11,500,287	11,610,994
SRR341725_1	243,597,814	245,539,614	243,816,854	243,815,254	64,208,192	63,670,486	63,976,622
SRR341725_2	256,207,141	258,341,321	256,477,481	256,475,521	67,798,935	67,211,453	67,540,655
SRR5853087_1	533,309,389	587,293,689	551,618,029	551,616,269	151,191,562	144,287,661	145,614,314
SRR957915_1	359,391,254	377,269,914	366,210,954	366,208,274	99,764,944	97,944,621	98,535,629
SRR957915_2	552,123,930	579,292,010	562,058,990	562,056,990	154,403,733	150,318,279	152,814,671
average	194,530,421	202,319,859	197,112,567	**197,111,258**	53,193,121	**52,007,543**	52,491,325

Table 2: For each dataset ($k = 21$) are reported the CL lower bound ((Rahman and Medvedev, 2020)), the CL and the compression for UST, USTAR and Eulertigs, representing the size of the sequence file compressed with MFCompress ((Pinho and Pratas, 2014)). Eulertigs achieved the optimum CL while USTAR obtained the best compression for almost all datasets.

compression, however the smallest CL does not guarantee to produce the better compression, since it might also depend on the number of paths/sequences.

k	**UST gap**	**USTAR gap**
15	42.58%	17.93%
17	16.23%	3.80%
21	3.09%	1.03%
31	1.57%	0.61%

Table 3: Average lower bound gap, over all datasets, varying $k \in \{15, 17, 21, 31\}$. The gap decreases with increasing k and UST strongly degrades its performance for lower k.

In all the following experiments we exclude Eulertigs because it does not support k-mer counts. For UST and USTAR we also assess the CL quality with respect to the lower bound provided by (Rahman and Medvedev, 2020). In Table 3 are reported the average gap, over all datasets, between the lower bound and the actual CL varying the k-mer size. We can observe that the UST gap is 1.57% for $k = 31$, in line with the results reported in Rahman and

Medvedev (2020), but for small values of k the performance worsens with a gap of 42.58% w.r.t. the lower bound. On the other hand, USTAR has consistently a lower gap, for all values of k, reaching 17.93% in the worst case. This confirms that k-mer sets are difficult to compress when k is small. However, USTAR obtains better performance w.r.t. UST, on the compression of k-mer sets.

3.2 Compression of k-mer Sets With Counts

In this section we tested a series of tools that are able to handle k-mers and counts. We selected the most popular k-mer counters and other compression tools as competitors: DSK, KMC, Squeakr, UST and BCALM. We ran all compression tools on all datasets for $k = 21$ and we reported the results in Table 4. In all cases, the stored data (sequences and counts) is additionally compressed with MFCompress (Pinho and Pratas, 2014) for nucleotide sequences and with bzip3 for binary or text data. In Table 4 are reported the total dimensions of the files compressed by the different tools.

We can observe that USTAR is on average the best compressor, and it consistently outperforms the other tools on all datasets. Traditional k-mer counters, like DSK, KMC and Squeakr, are not optimized for compression and they obtain poor compression performance. As expected UST and BCALM are the second and third best methods. However, USTAR produces a representation that is 76% smaller than BCALM and 4.2% over UST.

Since it is clear that UST is the best competitor among the tools that compress k-mers and counts, in the next tests we compare USTAR with UST. We used three different evaluation metrics:

- *CL*: the cumulative length as defined in section 2, to test the quality before the k-mers compression;
- *Counts*: the file size of counts after compression with a general-purpose

Dataset	DSK	KMC	Squeakr	BCALM	UST	USTAR
SRR001665_1	76,729,965	63,102,295	62,769,135	43,100,358	12,641,658	12,332,551
SRR001665_2	89,356,517	73,618,021	70,364,023	54,549,879	15,492,263	15,109,673
SRR061958_1	1,853,355,280	1,512,526,861	1,214,304,214	792,616,145	194,173,905	185,905,825
SRR061958_2	2,269,394,440	1,850,606,165	1,445,986,432	940,752,737	235,657,588	225,975,765
SRR062379_1	771,892,475	633,615,665	559,244,946	334,386,186	82,713,766	79,283,723
SRR062379_2	766,644,876	629,023,699	556,418,241	327,042,925	80,164,746	76,708,406
SRR10260779_1	594,043,132	489,620,438	459,620,233	272,605,742	64,644,700	61,724,139
SRR10260779_2	661,730,544	545,447,915	501,793,581	311,074,932	72,772,294	69,375,320
SRR11458718_1	660,336,575	547,192,385	515,587,875	278,247,157	64,694,925	61,236,404
SRR11458718_2	699,675,661	580,313,686	542,321,885	304,467,895	68,982,466	65,438,050
SRR13605073_1	286,147,403	236,056,529	244,522,615	110,324,321	25,833,347	24,546,244
SRR14005143_1	72,421,457	59,702,423	75,386,963	26,222,881	6,419,520	6,220,215
SRR14005143_2	148,413,200	121,547,826	126,063,532	51,976,493	13,117,896	12,655,430
SRR332538_1	61,647,503	50,466,675	65,343,140	21,192,576	5,737,778	5,599,034
SRR332538_2	125,336,255	100,440,228	116,698,603	77,057,667	14,410,775	13,528,977
SRR341725_1	972,617,730	799,134,833	700,565,933	262,398,076	80,436,678	78,193,253
SRR341725_2	1,005,087,513	825,643,578	719,993,731	277,159,709	84,250,689	81,877,574
SRR5853087_1	1,494,920,206	1,234,975,195	1,084,532,779	1,073,165,506	191,108,921	177,278,725
SRR957915_1	1,016,375,644	837,315,550	732,334,056	590,259,971	122,748,678	116,872,195
SRR957915_2	1,589,786,146	1,301,318,582	1,062,835,997	829,172,816	182,073,051	172,757,385
Average	760,795,626	624,583,427	542,834,396	348,888,699	80,903,782	**77,130,944**

Table 4: Datasets (with $k = 21$) are processed with DSK, KMC, Squeaker, BCALM, UST, and USTAR. Nucleotide files are then compressed with MFCompress ((Pinho and Pratas, 2014)) while other text or binary files are compressed with bzip3. The average file size over all datasets is reported in the last row.

compressor bzip3;

- *Overall*: the sum of compressed file sizes of k-mers and counts.

Given the metric M, we measure the improvement of USTAR over UST as

$$\Delta M = \frac{M_{UST} - M_{USTAR}}{M_{UST}} \tag{2}$$

where M can be CL, $Counts$ or $Overall$.

In the next experiment, we compared USTAR and UST with these three metrics. In Table 5 we reported the average improvements of USTAR w.r.t. UST when tested on all the above datasets while varying the size of k-mers, using odd lengths as previously done by other authors (Rahman and Medvedev, 2020; Rahman et al., 2021; Břinda et al., 2021).

From the table it is clear that for all values of k, USTAR obtained a greater compression ratio w.r.t. UST, for both nucleotide and count compression. For

***k*-mer size**	**ΔCL**[%]	**ΔCounts**[%]	**ΔOverall**[%]
15	33.64	12.07	26.40
17	13.61	15.85	13.92
21	2.10	14.17	4.20
31	0.97	12.70	2.30

Table 5: Average improvements of USTAR w.r.t. UST varying the k-mer size.

larger values of $k = 31$, USTAR demonstrates an overall advantage of 2.30%. This improvement primarily derives from compression of counts. Specifically, the difference in counts, ΔCounts, amounts to 12.70%.

The overall improvement increases for smaller values of the k-mer length $k = 21$ and $k = 17$ and it reaches the maximum value of 26.40% for $k = 15$. A similar behavior can be observed for ΔCL, which increases from 0.97% with $k = 31$ to 33.64% with $k = 15$. We can note that the count improvement is roughly constant as k varies, with an improvement of 12.07–15.85%. In general, ΔOverall is mainly driven by ΔCL.

Given USTAR's utilization of the de Bruijn graph's structure, it's interesting to explore its behavior concerning the number of arc. It's worth investigating how USTAR's performance correlates with the number of arcs in the graph. In the context of a dBG, nodes possess two sides, each potentially connected by four arcs (one per nucleotide). Consequently, the maximum number of arcs is calculated as $2 \cdot 4 \cdot \#nodes$. This insight allows us to define graph density as the ratio of the actual number of arcs to the maximum potential number of arcs:

$$density = \frac{\#arcs}{8 \cdot \#nodes}$$

In Figure 2 are shown the improvements in the cumulative length for all datasets with different k-mer sizes, plotted against their density (see Table 6 in the Appendix). Each point in the Figure represents a dataset. The plot has a well-defined curve that slowly raises until $density = 35\%$ and then increases

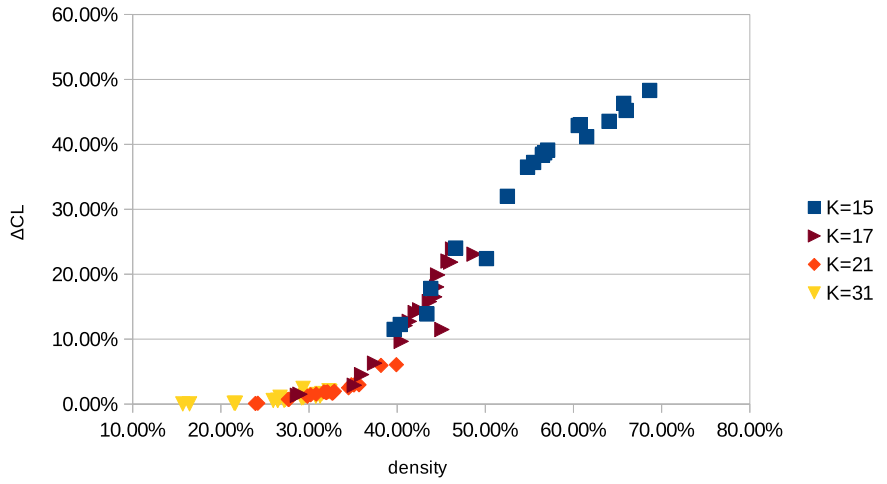


Figure 2: Enhancement on the cumulative length, w.r.t. UST, as a function of the dBG graph density. On the x-axis we have the density of all the datasets used with different k-mer length, i.e. $k \in \{15, 17, 21, 31\}$. On the y-axis we have the improvement of the cumulative length as defined in Equation 2. It's clear that ΔCL increases with the density of dBG suggesting us that the strategy applied by USTAR works better than choosing an arbitrary node for the path extension.

almost linearly. ΔCL spans from 0.02% to 48.31% showing that the compression ratio strongly increases as the density increases. We can observe that the graph density is very important in determining the improvement in k-mers compression because, with a denser graph, we obtained significantly higher improvements.

Another important property of the dBG graph is the number of isolated nodes. If a node is not connected to the rest of the graph it will be impossible to find a path that traverses the node, and thus it will be difficult to compress. We wanted to study the influence of isolated nodes on the compressibility of the dBG graph, by analyzing the CL improvement of USTAR w.r.t. UST. In Figure 3 are represented the CL as a function of the number of isolated nodes. It is clear that the datasets that have more isolated nodes obtained smaller CL: connected nodes can be reorganized wisely while isolated nodes do not

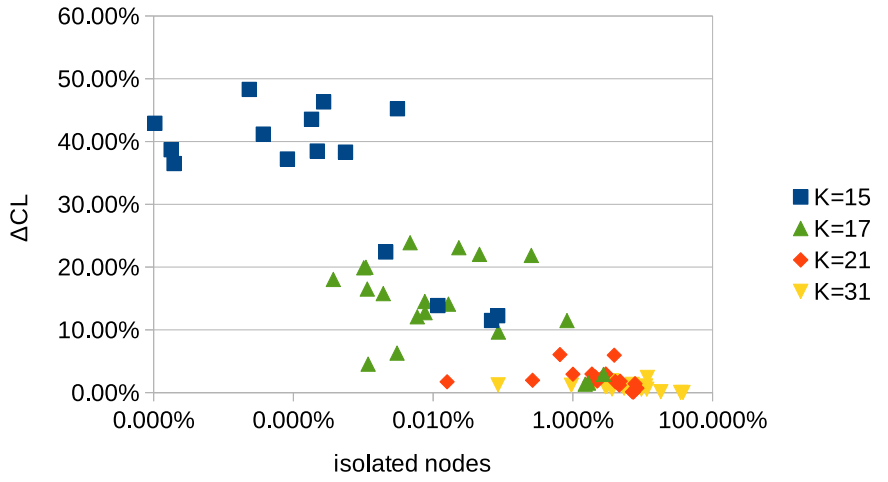


Figure 3: Enhancement on the cumulative length, w.r.t. UST, as a function of the isolated nodes in the dBG. On the x-axis we have the percentage of the isolated nodes of all the datasets used with different k-mer length, i.e. $k \in \{15, 17, 21, 31\}$. On the y-axis we have the improvement of the cumulative length as defined in Equation 2. The figure suggests that there exists a correlation between the number of isolated nodes and the improvement on CL.

offer better alternatives. Since lowering the k-mer size makes the graph more connected, due to the higher number of neighboring k-mers, a small k ensures a smaller number of isolated nodes and thus a better compressibility. We also considered other properties such as the number of k-mers, the number of unitigs, the read length, and the counts variance. However all these features are not as significant as the graph density and the isolated nodes, and so they are omitted.

Certainly, a denser graph provides USTAR with a larger pool of nodes to choose from compared to UST, which simply selects the first connected k-mer. This means that USTAR has more connected nodes available for constructing paths, potentially avoiding the creation of single-node paths. The higher compression ratio achieved by USTAR on denser graphs proofs the effectiveness of its strategy based on node connectivity. In contrast, UST's almost random path extension approach may not perform optimally, especially with denser

graphs. There's a possibility of generating numerous single-node paths, significantly increasing the number of paths and consequently the cumulative length of compressed k-mers, which worsens the compression ratio.

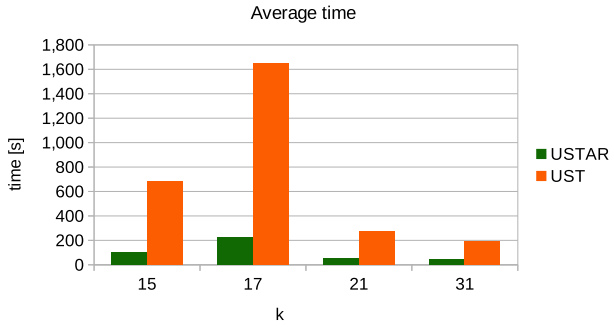
As we decrease the size of k-mers, we tend to observe higher graph density. With smaller k, there's a greater likelihood of having more connections between k-mers, leading to increased density.

In summary, USTAR demonstrates superior compression ratios for both k-mers and counts compared to various other tools. Its effectiveness is particularly pronounced on dense dBG graphs and for smaller k-mer sizes.

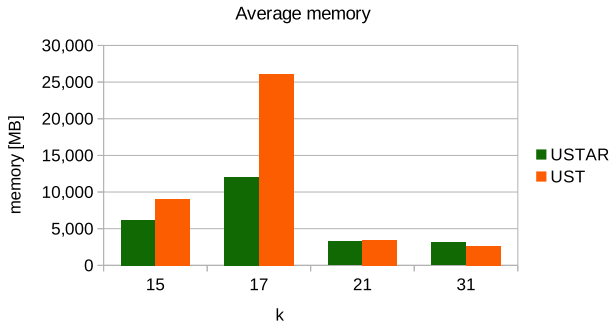
3.3 Time and Memory

In the preceding section, we established that UST and USTAR stand out as the best tools for compressing k-mer sets, considering k-mer counts. In this section we want to investigate potential performance disparities, in terms of running time and memory, between these two algorithms.

In Figure 4 are shown the average execution time and memory usage across different values of $k \in \{15, 17, 21, 31\}$. Generally, we observe elevated running times for smaller k values, reaching a peak for $k = 17$ before gradually decreasing with larger values of k. USTAR has proven to be considerably faster than UST, consistently outperforming it, reaching a maximum speedup of 7.2 for $k = 17$. The memory usage follows the same trend with USTAR using less memory for lower k-mer sizes and almost the same for bigger k. For $k = 17$, UST uses 2.17 times more memory than its competitor. Although both algorithms follow a similar strategy for traversing the graph, implementation choices and optimizations led USTAR to be much faster with a speedup between 4.2× and 7.2×.



(a)



(b)

Figure 4: Average running time and memory utilization of UST and USTAR varying the k-mer length. In Figure (a), USTAR time is considerably lower than UST, in particular, it can reach a maximum speedup of 7.22. In Figure (b) the memory used by USTAR is particularly low w.r.t. UST for $k = 17$ and $k = 15$.

4 Conclusions

This paper introduces USTAR, a tool designed for compressing sets of k-mers with counters. Our methodology involves employing a vertex-disjoint path cover to identify a representation of the k-mers set that minimizes the cumulative length of the compressed data. By strategically exploring de Bruijn graphs and making informed decisions based on node connectivity and average counts, we have achieved superior compression ratios compared to existing tools, including UST.

The evaluation of USTAR across various datasets, when compared to several other tools, consistently demonstrates its superiority, especially for smaller k-mer sizes. The observed improvements range from 0.97% to 33.64% in terms of cumulative length, remaining nearly constant for counts. The reduction in cumulative length is the driving force behind the overall improvement in compressed file size.

Moreover, our study reveals the significance of graph density in determining USTAR's effectiveness. Higher graph density, influenced by lower k-mer sizes, leads to greater improvements. The cumulative length reduction is influenced even by the number of isolated nodes, showcasing that USTAR is more effective when dealing with fewer of them.

Benchmarking results underscore USTAR as the fastest and most efficient tool for compressing sequences and counters, outperforming UST by up to 7.22 times in speed and 2.17 times in memory efficiency.

In conclusion, USTAR provides an effective solution for compressing k-mer sets with counters, leveraging de Bruijn graphs and strategic choices in constructing the vertex-disjoint path cover. The method's superior compression ratios and fast algorithm hold significant potential for enhancing storage and processing efficiency in bioinformatics analyses, facilitating more streamlined analysis of large-scale genomic data through k-mer-based tools.

Authors' Contributions

All authors conceived the study and drafted the manuscript. Enrico Rossignolo implemented the USTAR software and performed the experiments. All authors have read and approved the manuscript for publication.

Author Disclosure Statement

The authors declare they have no conflicting financial interests.

Funding Information

Authors are supported by the National Recovery and Resilience Plan (NRRP), National Biodiversity Future Center - NBFC, NextGenerationEU.

Acknowledgments

An earlier version of this paper appeared in the proceedings of the International Symposium on Bioinformatics Research and Applications (ISBRA), 2023, titled "Ustar: Improved compression of k-mer sets with counters using de bruijn graphs"(Rossignolo and Comin, 2023). We appreciate the feedback and suggestions received from reviewers and attendees of that conference.

References

Andreace, F., Pizzi, C., and Comin, M. Metaprob 2: Metagenomic reads binning based on assembly using minimizers and k-mers statistics. *Journal of Computational Biology*, 28(11):1052–1062, 2021. doi: 10.1089/cmb.2021.0270.

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.

Bradley, P., Den Bakker, H. C., Rocha, E. P., McVean, G., and Iqbal, Z. Ultra-

fast search of all deposited bacterial and viral genomic data. *Nature biotechnology*, 37(2):152–159, 2019.

Břinda, K., Baym, M., and Kucherov, G. Simplitigs as an efficient and scalable representation of de bruijn graphs. *Genome biology*, 22(1):1–24, 2021.

Cavattoni, M. and Comin, M. Classgraph: Improving metagenomic read classification with overlap graphs. *Journal of Computational Biology*, 30(6):633–647, 2023. doi: 10.1089/cmb.2022.0208. PMID: 37023405.

Chikhi, R., Limasset, A., and Medvedev, P. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12): i201–i208, 06 2016a. ISSN 1367-4803.

Chikhi, R., Limasset, A., and Medvedev, P. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016b.

Chikhi, R., Holub, J., and Medvedev, P. Data structures to represent a set of k-long dna sequences. *ACM Computing Surveys (CSUR)*, 54(1):1–22, 2021.

Conway, T. C. and Bromage, A. J. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–486, 01 2011. ISSN 1367-4803.

Denti, L., Previtali, M., Bernardini, G., Schönhuth, A., and Bonizzoni, P. Malva: genotyping by mapping-free allele detection of known variants. *Iscience*, 18:20–27, 2019.

Harris, R. S. and Medvedev, P. Improved representation of sequence bloom trees. *Bioinformatics*, 36(3):721–727, 2020.

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

Marchet, C., Iqbal, Z., Gautheret, D., Salson, M., and Chikhi, R. Reindeer: efficient indexing of k-mer presence and abundance in sequencing datasets. *Bioinformatics*, 36(Supplement_1):i177–i185, 2020.

Marcolin, M., Andreace, F., and Comin, M. Efficient k-mer indexing with application to mapping-free SNP genotyping. In Lorenz, R., Fred, A. L. N., and Gamboa, H., editors, *Proceedings of the 15th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2022, Volume 3: BIOINFORMATICS, February 9-11, 2022*, pages 62–70, 2022.

Monsu, M. and Comin, M. Fast alignment of reads to a variation graph with application to snp detection. *Journal of Integrative Bioinformatics*, 18(4): 20210032, 2021.

Ondov, B. D., Treangen, T. J., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S., and Phillippy, A. M. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17(1):1–14, 2016.

Pandey, P., Almodaresi, F., Bender, M. A., Ferdman, M., Johnson, R., and Patro, R. Mantis: a fast, small, and exact large-scale sequence-search index. *Cell systems*, 7(2):201–207, 2018a.

Pandey, P., Bender, M. A., Johnson, R., and Patro, R. Squeakr: an exact and approximate k-mer counting system. *Bioinformatics*, 34(4):568–575, 2018b.

Pinho, A. J. and Pratas, D. Mfcompress: a compression tool for fasta and multi-fasta data. *Bioinformatics*, 30(1):117–118, 2014.

Qian, J. and Comin, M. Metacon: Unsupervised clustering of metagenomic contigs with probabilistic k-mers statistics and coverage. *BMC Bioinformatics*, 20(367), 2019. doi: 10.1186/s12859-019-2904-4.

Rahman, A. and Medvedev, P. Representation of k-mer sets using spectrum-preserving string sets. In *International Conference on Research in Computational Molecular Biology*, pages 152–168. Springer, 2020.

Rahman, A., Chikhi, R., and Medvedev, P. Disk compression of k-mer sets. *Algorithms for Molecular Biology*, 16(1):1–14, 2021.

Rhie, A., Walenz, B. P., Koren, S., and Phillippy, A. M. Merqury: reference-free quality, completeness, and phasing assessment for genome assemblies. *Genome Biology*, 21:245, 2020.

Rizk, G., Lavenier, D., and Chikhi, R. Dsk: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, 2013.

Rossignolo, E. and Comin, M. Ustar: Improved compression of k-mer sets with counters using de bruijn graphs. In Guo, X., Mangul, S., Patterson, M., and Zelikovsky, A., editors, *Bioinformatics Research and Applications*, pages 202–213, Singapore, 2023. Springer Nature Singapore. ISBN 978-981-99-7074-2.

Schmidt, S. and Alanko, J. N. Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time. *bioRxiv*, pages 2022–05, 2022.

Storato, D. and Comin, M. K2mem: Discovering discriminative k-mers from sequencing data for metagenomic reads classification. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(1):220–229, 2022. doi: 10.1109/TCBB.2021.3117406.

Sun, C. and Medvedev, P. Toward fast and accurate snp genotyping from whole genome sequencing data for bedside diagnostics. *Bioinformatics*, 35(3):415–420, 2019.

Sun, C., Harris, R. S., Chikhi, R., and Medvedev, P. Allsome sequence bloom trees. *Journal of Computational Biology*, 25(5):467–479, 2018.

Wood, D. E. and Salzberg, S. L. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15(3):1–12, 2014.

Appendix

name	**#15-mers**	**#17-mers**	**#21-mers**	**#31-mers**
SRR001665_1	13,889,837	14,544,779	14,286,068	10,343,472
SRR001665_2	16,371,558	17,232,637	16,895,362	12,058,109
SRR061958_1	225,788,025	350,274,473	388,490,798	404,149,685
SRR061958_2	265,935,616	435,309,708	482,235,278	495,804,915
SRR062379_1	109,810,585	141,478,843	152,875,155	160,692,477
SRR062379_2	108,958,432	140,271,618	151,987,994	159,905,793
SRR10260779_1	84,250,397	104,694,175	113,667,728	123,624,245
SRR10260779_2	93,032,179	117,271,504	128,074,943	139,633,894
SRR11458718_1	89,998,269	115,532,020	126,431,861	137,995,280
SRR11458718_2	94,018,791	122,041,952	134,997,414	150,549,990
SRR13605073_1	43,488,336	51,381,500	54,085,000	55,764,573
SRR14005143_1	11,307,338	12,231,022	13,223,059	15,005,192
SRR14005143_2	23,691,810	26,274,190	28,456,533	31,850,681
SRR332538_1	10,624,064	11,122,375	11,404,027	11,382,816
SRR332538_2	18,741,106	21,970,819	25,674,930	28,880,136
SRR341725_1	132,442,790	180,354,754	188,913,254	185,618,107
SRR341725_2	136,484,353	186,965,667	196,035,961	192,133,588
SRR5853087_1	159,744,051	252,573,112	316,438,109	382,773,071
SRR957915_1	126,236,121	181,711,859	208,110,514	239,200,400
SRR957915_2	188,867,779	296,084,887	335,926,750	364,597,018

Table 6: Number of k-mers for each dataset varying $k \in \{15, 17, 21, 31\}$.

name	**k=15**	**k=17**	**k=21**	**k=31**
SRR001665_1	39.65%	28.68%	23.92%	15.68%
SRR001665_2	40.35%	28.97%	24.19%	16.45%
SRR061958_1	65.70%	45.77%	31.95%	26.44%
SRR061958_2	68.64%	46.29%	31.96%	25.94%
SRR062379_1	56.48%	42.05%	30.82%	27.17%
SRR062379_2	56.49%	42.55%	31.69%	27.31%
SRR10260779_1	54.79%	40.90%	32.10%	29.17%
SRR10260779_2	55.48%	41.37%	32.63%	29.35%
SRR11458718_1	56.73%	43.65%	34.48%	30.85%
SRR11458718_2	57.07%	44.25%	35.67%	32.37%
SRR13605073_1	52.50%	40.45%	29.81%	26.46%
SRR14005143_1	43.81%	35.96%	32.72%	31.26%
SRR14005143_2	46.63%	37.43%	32.87%	30.77%
SRR332538_1	43.37%	35.13%	30.19%	26.71%
SRR332538_2	50.13%	45.08%	38.16%	29.32%
SRR341725_1	60.54%	44.59%	27.70%	21.64%
SRR341725_2	60.78%	44.59%	27.64%	21.52%
SRR5853087_1	64.04%	48.72%	39.91%	32.27%
SRR957915_1	61.50%	44.47%	35.09%	31.45%
SRR957915_2	65.97%	46.07%	34.80%	29.58%

Table 7: Graph density of each dataset varying the k-mer size.

name	**k=15**	**k=17**	**k=21**	**k=31**
SRR001665_1	0.0686%	1.5032%	7.3743%	37.7526%
SRR001665_2	0.0841%	1.6698%	7.1247%	35.0048%
SRR061958_1	0.0003%	0.0461%	4.0547%	9.5807%
SRR061958_2	0.0000%	0.0047%	4.7626%	11.4403%
SRR062379_1	0.0006%	0.0166%	1.7678%	3.6429%
SRR062379_2	0.0002%	0.0076%	2.2492%	5.4160%
SRR10260779_1	0.0000%	0.0059%	1.8395%	2.9328%
SRR10260779_2	0.0001%	0.0077%	1.7530%	3.0750%
SRR11458718_1	0.0000%	0.0019%	2.0956%	3.9421%
SRR11458718_2	0.0000%	0.0011%	1.8775%	3.6168%
SRR13605073_1	0.0000%	0.0861%	4.6206%	6.5125%
SRR14005143_1	0.0000%	0.0012%	0.0159%	0.0853%
SRR14005143_2	0.0000%	0.0030%	0.2652%	0.9551%
SRR332538_1	0.0117%	2.7490%	7.7728%	11.0090%
SRR332538_2	0.0021%	0.8254%	3.9163%	11.6684%
SRR341725_1	0.0000%	0.0010%	8.2758%	18.1732%
SRR341725_2	0.0000%	0.0011%	8.0527%	18.3906%
SRR5853087_1	0.0002%	0.0234%	0.6544%	2.7970%
SRR957915_1	0.0000%	0.0004%	1.0035%	2.1041%
SRR957915_2	0.0031%	0.2544%	2.9701%	7.0781%

Table 8: The percentage of isolated nodes for each dataset varying the k-mer size.