USTAR: Improved Compression of *k*-mer Sets with Counters Using de Bruijn Graphs

Enrico Rossignolo and Matteo Comin(✉)

Department of Information Engineering, University of Padua, Padua, Italy
{enrico.rossignolo,matteo.comin}@unipd.it

Abstract. A fundamental operation in computational genomics is to reduce the input sequences to their constituent k-mers. Finding a space-efficient way to represent a set of k-mers is important for improving the scalability of bioinformatics analyses. One popular approach is to convert the set of k-mers into a de Bruijn graph and then find a compact representation of the graph through the smallest path cover.

In this paper, we present USTAR, a tool for compressing a set of k-mers and their counts. USTAR exploits the node connectivity and density of the de Bruijn graph enabling a more effective path selection for the construction of the path cover. We demonstrate the usefulness of USTAR in the compression of read datasets. USTAR can improve the compression of UST, the best algorithm, from 2.3% up to 26,4%, depending on the k-mer size.

The code of USTAR and the complete results are available at the repository https://github.com/enricorox/USTAR.

Keywords: k-mer set with counts · compression · smallest path cover

1 Introduction

The majority of bioinformatics analysis is performed by k-mer based tools that provide several advantages with respect to the ones that directly process reads or reads alignments. These tools operate primarily by transforming the input sequence data, which may be of various lengths depending on the technology used for sequencing, into a k-mer set that is a set of strings with fixed length and their multiplicities, called counts.

k-mers-based methods achieve better performance in many applications. In genome assembly, Spades [2] used k-mers-based methods to reconstruct the entire genome from reads obtaining efficiently highly accurate results. Also the assembly validation of Merqury [22] uses k-mer counts. In metagenomics, Kraken [27] is capable to classify and identify microorganisms in complex environmental samples using k-mers and it is 900 times faster than MegaBLAST. Since the introduction of Kraken, most of the tools for metagenomic classification are based on k-mers [1,5,19,24]. In genotyping, several tools [9,13,14,26] use k-mers instead of alignment to identify genetic variations in individuals or populations. In phylogenomics, Mash [15] uses k-mers to efficiently estimate genomes and metagenomes

X. Guo et al. (Eds.): ISBRA 2023, LNBI 14248, pp. 202–213, 2023.
https://doi.org/10.1007/978-981-99-7074-2_16

distances in order to reconstruct evolutionary relationships among organisms. In database searching many k-mers-based methods [3,10,12,16,25] have been proposed in order to search sequences efficiently.

Overall, k-mer-based methods have revolutionized many areas of bioinformatics and they have become an essential tool for analyzing large-scale genomic data. These tools often rely on specialized data structures for representing sets of k-mers (for a survey, see [6]). Since modern sequencing datasets are huge, the space used by such data structures is a bottleneck when attempting to scale up to large databases. Conway and Bromage [8] showed that at least $\log \binom{4^k}{n}$ bits are needed to losslessly store a set of n k-mers, in the worst case. However, a set of k-mers generated from a sequencing experiment typically exhibits the spectrum-like property [6] and contains a lot of redundant information. Therefore, in practice, most data structures can substantially improve on that bound [7].

Since storing a k-mer set requires non-negligible space, it's desirable to reduce the size that can be very large. For example, the dataset used to test the BIGSI [21] index takes approximately 12 TB to be stored in compressed form.

The best tool to compress a set of k-mers with counts is UST [21] (see Sect. 1.1) and it uses the De Bruijn graph representation of the input k-mer set. The problem of finding the smallest k-mer set representation is equivalent to finding the smallest path cover in a de Bruijn graph (Sect. 2). In this paper, we present USTAR (Unitig STitch Advanced constRuction), which follows a similar paradigm, but implements a better strategy for exploring De Bruijn graphs. The USTAR strategy leverages the density of the de Bruijn graph and node connectivity, enabling a more effective path selection for the construction of the path cover, and thus improving the compression. In Sect. 3 we reported a series of results on several real sequencing datasets. We showed that USTAR achieves the best compression ratio of k-mers and counts and it outperforms UST, and other tools.

1.1 Related Works

The problem of k-mer set compression has been addressed by several researchers, in this section we summarize the most recent findings. k-mers counters are tools that are designed to count and store distinct k-mers, a particularly hard challenge for large datasets. The most famous tools are Squeakr [17], KMC [11] and DSK [23]. Squeakr is an approximate and exact k-mers counting system that exploits Bloom filters, a probabilistic data structure, in order to efficiently store k-mers. KMC uses disk files as bins in which divide, sort, and count k-mers. Finally, DSK uses hash tables in order to update k-mers counters. The last two tools are not specifically designed for compression but they are still capable to reduce the size storing only distinct k-mers and their counts.

A k-mer set with counters can be represented by a de Bruijn graph (dBG) that can be exploited for efficient storage. BCALM2 [7] is a tool for the low-memory construction of dBGs that are compacted, meaning that maximal non-branching paths are merged in a single node labeled with k-mers glued together

and the list of the counts. Compaction not only provides advantages in terms of memory used but also in terms of disk space. The idea, in order to compress k-mers, is to save $k-1$ characters per link. For example, given a dBG, the non-branching path (ACT, CTG, TGA) can be replaced by a single node $ACTGA$. The sequence represented by a non-branching path is called unitig and it is an attempt to compress k-mers using dBGs.

Another way to reduce the redundancy of a k-mer set is to exploit its spectrum-like property [6], i.e. the existence of long strings that "generate" all the k-mers. This idea has been developed in parallel by the authors of ProphAsm [4] and UST [21]. The authors of ProphAsm [4] refer to these long strings as simplitigs and they build them by linking overlapping unitigs and k-mers during the exploration of a dBG computed on the fly. They showed that simplitigs outperformed unitigs, the k-mers representation proposed by BCALM2, in terms of computational resources and compression rate. Also UST [21] links overlapping unitigs and k-mers, but it uses as input the compacted de Bruijn graph computed by BCALM2 and it considers also k-mers counts. They find a nearly tight lower bound for the optimal k-mers representation and they showed empirically that in most cases their greedy algorithm is within 3% of the lower bound.

2 USTAR: Unitig STitch Advanced ConstRuction

2.1 Definitions

For the purpose of this paper, we consider a string made up of characters in $\Sigma = \{A, C, T, G\}$. A string of length k is called k-mer and its *reverse complement* $rc(\cdot)$ is obtained by reversing the k-mer and replacing each character with its complement, that is $A \mapsto T$, $C \mapsto G$, $T \mapsto A$, $G \mapsto C$. Since we don't know from which DNA strand it is taken, we consider a k-mer and its reverse complement as the same k-mer. Given a string $s = \langle s_1, \ldots, s_{|s|} \rangle$, we denote the first i characters of s as $pref_i(s) = \langle s_1, \ldots, s_i \rangle$ and the last i characters of s as $suf_i(s) = \langle s_{|s|-i+1}, \ldots, s_{|s|} \rangle$. We define the *glue* operation between two strings u and v such that $suf_{k-1}(u) = pref_{k-1}(v)$ as the concatenation of u and the suffix of v:

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

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

A set of k-mers can be represented by a de Bruijn graph, of which we will give a node-centric definition, meaning that the arcs are implicitly given by the nodes. Thus we can refer to k-mers and dBG(K) interchangeably.

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

1. $V = \{v_1, \ldots, v_{|K|}\}$
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 [21].

Note that nodes' sides allow treating a k-mer and its reverse complement as if they were the same k-mer. Furthermore, nodes can be associated with k-mer counts.

A path $p = \langle (v_1, s_1), \ldots, (v_l, s_l) \rangle$ is spelled by gluing the spelling of its node sides:

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

The path p is said to be a *unitig* if its internal nodes have in-degree and out-degree equal to 1. A unitig is said to be *maximal* if it cannot be extended on either side. In order to decrease its memory footprint, a dBG(K) can be *compacted* by replacing maximal unitigs with single nodes labeled with the spellings of the unitigs.

An example of compacted dBG(K) with

$$K = \{ACT, CTG, TGA, CTT, TTG, TGC\}$$

is in Fig. 1. It has been compacted by replacing the maximal unitig (CTT, TTG) with the node $CTTG$.

2.2 Vertex-Disjoint Path Cover Problem

Compressing a k-mer set K can be achieved by finding a representation S of K made of strings of any length such that the set of its substrings of length k is equal to K.

A natural way to measure the size of a string set S is by computing its *cumulative length* defined as the sum of all the string lengths:

$$CL(S) = \sum_{s \in S} |s|$$

where $|s|$ is the length of the string s. It has been shown in [4,21] that, when S does not contain duplicate k-mers, the cumulative length of S is proportional to its cardinality, in particular, it holds

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

where $|S|$ is the cardinality of the set S. Therefore our goal, finding the best representation of a k-mer set K, is equivalent to minimizing the number of strings in the string set S.

Consider again the example in Fig. 1. From the path $p = (ACT, CTG, TGA)$ we can compute its spell $spell(p) = ACTGA$ that contains all the 3-mers ACT, CTG and TGA in p. Thus from a set of paths P that contains all the nodes in dBG(K) we can derive a set S of compressed k-mers. By imposing that all the paths are vertex-disjoint, we guarantee that k-mers are represented only once. Therefore a vertex-disjoint path cover can be used in order to compute S for compression.

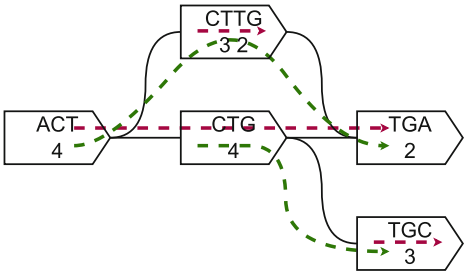

Fig. 1. An example of a compacted de Bruijn graph. Nodes are labeled with k-mers and their counts. Undirected arcs are used in place of two arcs with opposite directions. UST may choose the path cover in red while USTAR is forced to choose the path cover in green. (Color figure online)

Recalling that in order to minimize the cumulative length of a string set S that represents K we need to minimize the number of strings $|S|$ and that it corresponds to the number of paths $|P|$, we can solve the following problem.

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.

For general graphs the problem above is known to be NP-hard [4,21] since it can be reduced from Hamilton: a graph has a Hamiltonian path if and only if it has a vertex-disjoint path cover of cardinality 1. However, it is not clear if the problem is still NP-hard for de Bruijn graphs.

Nevertheless, greedy and non-optimal algorithms have been proposed. ProphAsm [4] 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 [21] 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. If we consider the example in Fig. 1, ProphAsm

and UST, by choosing nodes arbitrarily, may build the path cover (in red) $P = \{(ACT, CTG, TGA), (CTT, TTG), (TGC)\}$ from which derives, by computing the spelling of each path, the set of strings $S = \{ACTGA, CTTG, TGC\}$. In this example, the cumulative length is $CL(S) = 12$.

In this work, we present USTAR (Unitig STitch Advanced constRuction) that, unlike previous algorithms, exploits the connectivity of the dBG graph and the values of counts. USTAR also implements a heuristic to compute simplitigs. As UST, also USTAR takes advantage of the compacted de Bruijn graph computed by BCALM2. Similarly to UST and ProphAsm, at each step, USTAR selects a seed node in the graph, and then it tries to compute a path starting from this node. A path is constructed by connecting adjacent nodes until the path cannot be further extended. The algorithm continues with the selection of a new seed node until all nodes have been covered by a path. The two key operations in this algorithm are how to select a good seed node, and how to extend a path among the available connections.

In USTAR the counts associated with each node and the topology properties can be used to determine the best seed and how to extend it. The distribution of counts is in general very skewed, with several low values and few very high values. Since the counts distribution is non-uniform, and skewed, it turns out that higher counts are harder to compress. For this reason, the exploration strategy of USTAR chooses as seed the node that has the highest average counts. In general neighboring nodes usually have similar counts, so that choosing the seed based on the highest average count might improve the compression of these high counts.

As for the path cover construction, we observe that UST and ProphAsm might choose a highly connected node, and since this node will not be available in the subsequent iterations, this selection may lead to isolated nodes, that will increase the cumulative length, like in the example in Fig. 1. Instead, in USTAR we try to avoid this scenario and, in fact, paths are extended by selecting the node with fewer connections so that highly connected nodes are still available for future iterations. These choices will help to have a lower CL since they create fewer and longer simplitigs.

Following the example in Fig. 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 $CL(S') = 10$.

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

Overall, in this example, the uncompressed k-mer set will require $CL(K) = 18$, with UST the k-mer set can be compressed with $CL(S) = 12$, whereas USTAR will produce a better compression with $CL(S') = 10$.

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, KMC, DSK, BCALM2, and UST. We used for testing a set of real reads datasets taken from previous studies [4,7,11,17,23]. A summary of all datasets is reported in Table 1. For each dataset, we extracted all k-mers (see Table 1) and the corresponding counts, and use this information as input for all compression tools. For some tools, like UST and USTAR, it is required to build a compacted dBG with BCALM2 as a preprocessing.

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

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	2 5,479,128	1.254
SRR5853087	Danio rerio RNA-Seq	101	119,482,078	3.194
SRR957915	Human RNA-seq	101	49,459,840	3.671

In the first experiment, we ran all compression tools on all datasets for $k = 21$ and we reported the results in Table 2. In all cases, the stored data is additionally compressed using MFCompress [18] for nucleotide sequences or with bzip3 for binary data. In Table 2 are reported the 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. As expected UST and BCALM are the second and third best methods, however, USTAR shrinks the representation by 76% over BCALM and 4.2% over UST.

Since it is clear that UST is the best competitor in the next tests we compare USTAR with UST. We used three different evaluation metrics:

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

Table 2. Datasets (with $k = 21$) are processed with DSK, KMC, Squeaker, BCALM, UST, and USTAR. Nucleotide files are then compressed with MFCompress [18] while other text or binary files are compressed with bzip3. The average file size over all datasets is reported in the last row.

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

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

Given the metric M the improvement over UST is computed as

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

where M can be CL, *counts* or *overall*.

In the next experiment, we compared USTAR and UST with these three metrics. In Table 3 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 [4,20,21].

We can observe that for all values of k, USTAR improves over UST, for both nucleotide and count compression. For large values of $k = 31$, the overall advantage of USTAR is 2.30%, and this improvement is mainly achieved with a better compression of counts, in fact, Δcounts = 12.70%. We suspect that with higher values of k the compression might decrease, but more experiments are needed to confirm this hypothesis.

If smaller values of k are considered the overall improvement increases with $k = 21$ and $k = 17$ and it reaches the maximum value of 26.40% for $k = 15$.

Table 3. Average improvements of USTAR w.r.t. UST varying the k-mer size.

k-mer size	Δ CL [%]	Δ counts [%]	Δ overall [%]
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

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.

Since USTAR exploits the structure and connectivity of the dBG graph it is interesting to further study the behavior of USTAR w.r.t. to the number of arcs in the graph. Recall that nodes in a dBG have two sides and that each side can have four arcs, one per nucleotide, thus the maximum number of arcs is $2 \cdot 4 \cdot \#nodes$. Based on this observation we can define the graph density as the number of arcs over the maximum number of arcs:

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

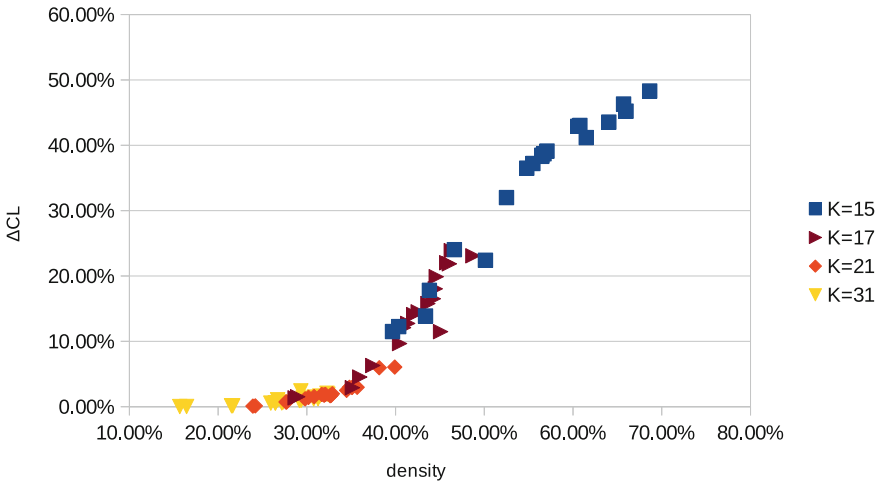

Fig. 2. Improvement on the cumulative length, w.r.t. UST, as a function of the dBG graph density.

In Fig. 2 are shown the improvements in the cumulative length for all datasets with different k-mer sizes, plotted against their density. Each point in the Figure

represents a dataset. The plot has a well-defined curve that slowly raises until $density = 35\%$ and then increases almost linearly. ΔCL spans from 0.02% to 48.31% showing that the compression ratio strongly increases as the density increases. We can say 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. We also considered other properties such as the number of k-mers, the number of unitigs, the number of isolated nodes, the read length, and the counts variance. However all these features are not as significant as the graph density, and so they are omitted.

Indeed, a higher density graph implies that USTAR has more nodes from which it can choose, unlike UST which picks the first connected k-mer, so that more connected nodes can be available to paths that would have been made up of single nodes. The higher compression ratio achieved by USTAR on denser graphs confirms that the strategy based on node connectivity works well. The fact that UST chooses almost randomly how to extend a path, it cannot guarantee to work well, even with a denser graph there exists the possibility that many single-node paths are generated, which substantially increases the number of paths and thus the cumulative length of compressed k-mers, worsening the compression ratio. If we decrease the k-mer size we observe a higher density: with small k there are more k-mers and it is more likely to have more connections between k-mers leading to an increase in density. In summary, USTAR can compress k-mers and counts with a better compression ratio w.r.t. to several other tools. USTAR is more effective on dense dBG graphs and for small k-mer sizes.

4 Conclusions

In this paper, we have presented USTAR, a tool for compressing k-mer sets with counters. Our approach utilizes a vertex-disjoint path cover to find a representation of the k-mer set that minimizes the cumulative length of the compressed data. By exploring de Bruijn graphs and making informed choices based on node connectivity and average counts, we have achieved better compression ratios compared to existing tools such as UST.

We have evaluated USTAR using various datasets and compared it with several other tools. The results demonstrate that our method consistently outperforms UST, and other tools, in terms of compression ratio, especially for smaller k-mer sizes. The improvements range from 0.97% to 33.64% in terms of cumulative length and it's almost constant for counts. The overall improvement in compressed file size is driven by the reduction in cumulative length.

Furthermore, we have observed that graph density plays a crucial role in determining the effectiveness of USTAR. Denser graphs yield higher improvements, and lowering the k-mer size contributes to increased density. These findings highlight the power of our method, particularly for datasets with higher graph density.

In conclusion, USTAR offers an effective solution for compressing k-mer sets with counters. By leveraging de Bruijn graphs and making informed choices in

the construction of the vertex-disjoint path cover, we have achieved superior compression ratios compared to existing tools. Our method has the potential to enhance the storage and processing efficiency of analysis in bioinformatics, enabling more efficient analysis of large-scale genomic data using k-mer-based tools.

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

References

1. Andreace, F., Pizzi, C., Comin, M.: Metaprob 2: metagenomic reads binning based on assembly using minimizers and k-mers statistics. J. Comput. Biol. **28**(11), 1052–1062 (2021). https://doi.org/10.1089/cmb.2021.0270
2. Bankevich, A., et al.: Spades: a new genome assembly algorithm and its applications to single-cell sequencing. J. Comput. Biol. **19**(5), 455–477 (2012)
3. Bradley, P., Den Bakker, H.C., Rocha, E.P., McVean, G., Iqbal, Z.: Ultrafast search of all deposited bacterial and viral genomic data. Nat. Biotechnol. **37**(2), 152–159 (2019)
4. Břinda, K., Baym, M., Kucherov, G.: Simplitigs as an efficient and scalable representation of de Bruijn graphs. Genome Biol. **22**(1), 1–24 (2021)
5. Cavattoni, M., Comin, M.: Classgraph: improving metagenomic read classification with overlap graphs. J. Comput. Biol. **30**(6), 633–647 (2023). https://doi.org/10.1089/cmb.2022.0208, pMID: 37023405
6. Chikhi, R., Holub, J., Medvedev, P.: Data structures to represent a set of k-long DNA sequences. ACM Comput. Surv. (CSUR) **54**(1), 1–22 (2021)
7. Chikhi, R., Limasset, A., Medvedev, P.: Compacting de Bruijn graphs from sequencing data quickly and in low memory. Bioinformatics **32**(12), i201–i208 (2016)
8. Conway, T.C., Bromage, A.J.: Succinct data structures for assembling large genomes. Bioinformatics **27**(4), 479–486 (2011)
9. Denti, L., Previtali, M., Bernardini, G., Schönhuth, A., Bonizzoni, P.: Malva: genotyping by mapping-free allele detection of known variants. Iscience **18**, 20–27 (2019)
10. Harris, R.S., Medvedev, P.: Improved representation of sequence bloom trees. Bioinformatics **36**(3), 721–727 (2020)
11. Kokot, M., Długosz, M., Deorowicz, S.: KMC 3: counting and manipulating k-mer statistics. Bioinformatics **33**(17), 2759–2761 (2017)
12. Marchet, C., Iqbal, Z., Gautheret, D., Salson, M., Chikhi, R.: Reindeer: efficient indexing of k-mer presence and abundance in sequencing datasets. Bioinformatics **36**(Supplement_1), i177–i185 (2020)
13. Marcolin, M., Andreace, F., Comin, M.: Efficient k-mer indexing with application to mapping-free SNP genotyping. In: Lorenz, R., Fred, A.L.N., Gamboa, H. (eds.) Proceedings of the 15th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2022, Volume 3: BIOINFORMATICS, 9–11 February 2022, pp. 62–70 (2022)
14. Monsu, M., Comin, M.: Fast alignment of reads to a variation graph with application to SNP detection. J. Integr. Bioinform. **18**(4), 20210032 (2021)
15. Ondov, B.D., et al.: Mash: fast genome and metagenome distance estimation using minhash. Genome Biol. **17**(1), 1–14 (2016)

16. Pandey, P., Almodaresi, F., Bender, M.A., Ferdman, M., Johnson, R., Patro, R.: Mantis: a fast, small, and exact large-scale sequence-search index. Cell Syst. **7**(2), 201–207 (2018)
17. Pandey, P., Bender, M.A., Johnson, R., Patro, R.: Squeakr: an exact and approximate k-mer counting system. Bioinformatics **34**(4), 568–575 (2018)
18. Pinho, A.J., Pratas, D.: Mfcompress: a compression tool for fasta and multi-fasta data. Bioinformatics **30**(1), 117–118 (2014)
19. Qian, J., Comin, M.: Metacon: unsupervised clustering of metagenomic contigs with probabilistic k-mers statistics and coverage. BMC Bioinform. **20**(367) (2019). https://doi.org/10.1186/s12859-019-2904-4
20. Rahman, A., Chikhi, R., Medvedev, P.: Disk compression of k-mer sets. Algorithms Mol. Biol. **16**(1), 1–14 (2021)
21. Rahman, A., Medvedev, P.: Representation of *k*-mer sets using spectrum-preserving string sets. In: Schwartz, R. (ed.) RECOMB 2020. LNCS, vol. 12074, pp. 152–168. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45257-5_10
22. Rhie, A., Walenz, B.P., Koren, S., Phillippy, A.M.: Merqury: reference-free quality, completeness, and phasing assessment for genome assemblies. Genome Biol. **21**, 245 (2020)
23. Rizk, G., Lavenier, D., Chikhi, R.: DSK: k-mer counting with very low memory usage. Bioinformatics **29**(5), 652–653 (2013)
24. Storato, D., Comin, M.: K2mem: discovering discriminative k-mers from sequencing data for metagenomic reads classification. IEEE/ACM Trans. Comput. Biol. Bioinf. **19**(1), 220–229 (2022). https://doi.org/10.1109/TCBB.2021.3117406
25. Sun, C., Harris, R.S., Chikhi, R., Medvedev, P.: Allsome sequence bloom trees. J. Comput. Biol. **25**(5), 467–479 (2018)
26. Sun, C., Medvedev, P.: Toward fast and accurate SNP genotyping from whole genome sequencing data for bedside diagnostics. Bioinformatics **35**(3), 415–420 (2019)
27. Wood, D.E., Salzberg, S.L.: Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biol. **15**(3), 1–12 (2014)