A linear algorithm for efficient representation of *k*-mer sets using de Bruijn graphs

Enrico Rossignolo and Matteo Comin

Department of Information Engineering, University of Padova, Padova, 35131,Italy
{enrico.rossignolo,matteo.comin}@unipd.it

Abstract. A fundamental operation in computational genomics is the reduction of input sequences into their constituent k-mers. Developing space-efficient methods to represent a collection of k-mers is crucial for enhancing the scalability of bioinformatics analyses. A common strategy is to transform the set of k-mers into a de Bruijn graph and then create a streamlined representation by identifying the smallest path cover.
In this article, we introduce USTAR2, a novel algorithm for compressing k-mers. USTAR2 leverages node connectivity principles in the de Bruijn graph for more efficient path selection in constructing the path cover. We tested USTAR2 on real read datasets and compared it with several other tools. USTAR2 demonstrated superior performance in terms of compression, requiring less memory and being significantly faster (up to 96x).
The code of USTAR2 is available at the repository `https://github.com/CominLab/USTAR2`

Keywords: k-mer set; compression; Bruijn graphs.

1 Introduction

The field of computational genomics significantly benefits from the utilization of k-mer-based tools, which present numerous advantages over methodologies that directly process reads or read alignments. These tools primarily work by converting input sequence data, which can vary in length according to the sequencing technology employed, into a collection of k-mers—fixed-length sequences—accompanied by their respective counts. This conversion facilitates a more streamlined and effective analysis of genomic data.

k-mer-based methods have always exhibited superior performance across a range of applications. For instance, in genome assembly, tools such as Spades [2] utilize k-mer-based techniques to reconstruct entire genomes from reads with high accuracy.

In metagenomics, Kraken [35] excels at classifying microorganisms in complex environmental samples using k-mers, offering a speedup of 900× compared to MegaBLAST. As a result, many tools for metagenomic classification are now k-mer-based [1,23,5,32]. In genotyping, several tools [9,34,17,18] utilize k-mers

instead of alignments to identify genetic variations in individuals and populations. In phylogenomics, Mash [19] employs k-mers to estimate distances between genomes and metagenomes, facilitating the reconstruction of evolutionary relationships among organisms. In database searching, a wide array of k-mer-based methods [33,11,3,20,16] have been developed to efficiently search sequences. k-mer-based methods have revolutionized multiple aspects of bioinformatics and have become indispensable for analyzing large-scale genomic data. To handle the vast modern sequencing datasets, these tools often rely on specialized and efficient data structures for representing sets of k-mers.

Storing sets of k-mers can be space-intensive, particularly for large databases. Conway and Bromage [8] found that, in the worst case, at least $\binom{\log 4^k}{n}$ bits are required to losslessly store a set of n k-mers. However, k-mer sets derived from sequencing experiments often exhibit a spectrum-like property [6] and contain many redundant information. Consequently, efficient data structures can significantly optimize this storage requirement [7].

Given the substantial space requirements for storing k-mer sets, reducing their size is essential. For instance, the dataset used to evaluate the BIGSI [24] index occupies roughly 12 TB of storage even in compressed form.

Given the vast volume of data accessible to bioinformaticians, the need for efficient data representation becomes imperative. With the ongoing advancements in sequencing technologies, the amount of genomic data is expected to grow substantially in the coming years. Therefore, analyzing this data will increasingly require serious computational resources.

Nevertheless, this challenge can be mitigated by using a solid k-mers representation that reduces RAM usage and accelerates analysis tools, allowing for the execution of more extensive pipelines with less computational effort. To achieve this objective, a compact, plain text representation has emerged as the most reasonable solution.

Formally, a plain text representation embodies a set of strings that includes every k-mer derived from the input sequences, whether in their forward or reverse-complemented forms, while excluding any other k-mers. This set, which may contain k-mer repetitions, is referred to as a Spectrum-Preserving String Set (SPSS), according to Schmidt et al [30]. It is noteworthy that this definition diverges from the one proposed by Rahman and Medvedev [24], who additionally require that each k-mer appears at most once within the set.

A plain text representation offers significant advantages in certain tools, such as Bifrost's query [12], which can utilized a plain text representation as a index without requiring any modifications. Furthermore, a plain text form does not require decompression that causes overhead in the entire pipeline.

In this paper, we introduce USTAR2 [1], an algorithm for k-mer set compression that leverages the correspondence between a path cover in a De Bruijn graph and its k-mer set representation. The tool exploits the graph's node connectivity and intelligently reuses nodes to minimize the impact of the representation.

[1] A preliminary version of USTAR2 has been presented at BIOINFORMATICS 2024 [28].

1.1 Related works

Rahman and Medvedev [24], along with Břinda, Baym, and Kucherov [4], independently explored the concept of storing a set of k-mers in plain text without repetitions to achieve a more compact representation. They named this concept respectively "Spectrum-Preserving String Set" (SPSS) [24] and "simplitigs" [4]. To avoid confusion with the new definition of SPSS, we refer to this concept as simplitigs.

Both Rahman and Medvedev with UST and Břinda, Baym, and Kucherov with ProphAsm propose algorithms that involve greedily joining consecutive unitigs to create such a representation. UST operates on the node-centric de Bruijn graph constructed from the input strings. It identifies arbitrary paths within this graph, starting from arbitrary nodes. Each node is visited by one path, and if a path cannot be extended forward, either due to a dead-end or all successor nodes having been visited, a new path begins from a new node. If, before starting a new path, any successor node of the finished path denotes the start of a different path, these two paths are merged. During traversal, the unitigs of the visited nodes are concatenated, ensuring that k-1 overlapping characters are not repeated. These concatenated strings constitute the final output.

ProphAsm does not construct a de Bruijn graph. Instead, the approach involves collecting all k-mers into a hash table and then extending each k-mer in both forward and backward directions as far as possible without repetition.

These heuristic approaches significantly minimize the number of strings (string count, SC) and the total length of these strings (cumulative length, CL) required to store a k-mer set. The reduction in CL directly leads to lower memory usage for string storage. Additionally, a decrease in SC is beneficial as it reduces the size of the index structure needed to store the strings, making the overall storage more efficient.

Břinda, Baym, and Kucherov showed that using simplitigs significantly reduces both the SC and CL in tangled de Bruijn graphs, such as those derived from single large genomes with short k-mer lengths and pangenome graphs with many genomes. They highlighted the benefits of simplitigs in downstream applications, like faster k-mer query run times with BWA [15]. Moreover, they found that storing simplitigs with a general-purpose compressors requires less space compared to compressed unitigs.

Similarly, the authors of UST reported substantial reductions in SC and CL across various datasets and noted that simplitigs occupy less space than unitigs when compressed.

Additionally, Khan et al. [13] provided a comprehensive overview of simplitigs applications to various genomic datasets, including a human gut metagenome.

Both the authors of UST and ProphAsm established a lower bound on the cumulative length of simplitigs and demonstrated that their heuristics produce representations with a cumulative length close to this bound for typical k values, such as 31. However, for smaller k values (e.g., < 20), which result in denser de Bruijn graphs, their heuristics do not approach the lower bound as closely.

Recently, a new heuristic called USTAR (Unitig STitch Advanced constRuction) was proposed in [26,27]. USTAR employs a strategy based on graph connectivity to explore de Bruijn graphs more effectively. By leveraging the density of the de Bruijn graph and node connectivity, USTAR enhances path selection for constructing the path cover, resulting in better compression, especially for denser de Bruijn graphs.

Several existing tools utilize simplitigs. For example, Cuttlefish2, a compacted de Bruijn graph builder [13], includes functionality to output simplitigs rather than maximal unitigs. Additionally, there has been a recent proposal for a standardized file format for k-mer sets that specifically supports simplitigs [10] and other plain text representations.

All these authors have investigated whether computing minimum simplitigs without repeating k-mers might be NP-hard. However, Schmidt and Alanko [31] recently proved that simplitigs with minimum cumulative length, named "eulertigs", can be computed in polynomial time. Their algorithm constructs a bidirected arc-centric de Bruijn graph in linear time using a suffix tree, followed by eulerization and the computation of a bidirected Eulerian circuit in the eulerized graph. Eulertigs optimally represent simplitigs by concatenating consecutive unitigs without repeating k-mers.

Interestingly, Eulertigs are only slightly smaller than the strings produced by previous heuristics, indicating that significant further improvements may be limited when k-mer repetitions are not allowed in a plain text representation.

In a recent study [30], the authors introduced the first algorithm for finding an SPSS of minimum size (CL) that allows for repeated k-mers. The compression advantage of SPSS compared to simplitigs and Eulertigs is significant. They demonstrated that finding a minimum SPSS with repeated k-mers is polynomially solvable, utilizing a many-to-many min-cost path query and a min-cost perfect matching approach. However, this optimal algorithm, called Matchtigs, requires $O(n^3m)$ time, where n is the number of nodes and m is the number of arcs in the de Bruijn graph, making it impractical for large datasets.

To address this, the same authors proposed a greedy heuristic for generating a compact SPSS, called Greedy Matchtigs, which gives up the optimal matching to improve efficiency.

In the next sections, we introduce USTAR2, a faster and more memory-efficient greedy heuristic designed to generate a compact SPSS for large datasets. USTAR2 is based on the USTAR paradigm [26,27], but it allows for repeated k-mers and can explore the de Bruijn graph more deeply, resulting in improved compression.

2 USTAR2: Unitig STitch Advanced constRuction 2

2.1 Definitions

In the context of this paper, we consider a string composed of characters from the set $\Sigma = \{A, C, T, G\}$. A string with a length of k is referred to as a "k-mer".

Its "reverse complement", denoted as $rc(\cdot)$, is derived by reversing the k-mer and substituting each character with its complementary base, such as $A \mapsto T$, $C \mapsto G$, $T \mapsto A$, and $G \mapsto C$. Since the origin DNA strand is unknown, we treat a k-mer and its reverse complement as identical.

Given a string $s = \langle s_1, \ldots, s_{|s|} \rangle$, we use $pref_i(s)$ to denote the first i characters of s, and $suf_i(s)$ for the last i characters. We introduce the "glue" operation between two strings, u and v, where $suf_{k-1}(u)$ matches $pref_{k-1}(v)$. This operation concatenates u with 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 results in $u \odot^2 v = CTGA$.

A collection of k-mers can be graphically represented by a de Bruijn graph, where we introduce a node-centric definition, indicating that the connections (arcs) are implicitly defined by the nodes. Therefore, we use the terms *k-mers set* and *dBG(K)* interchangeably.

For a given set of k-mers $K = \{m_1, \ldots m_{|K|}\}$, a de Bruijn graph of K is a directed graph, $\mathrm{dBG}(K) = (V, A)$, with the following attributes:

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 distinct sides $s_v \in \{0, 1\}$, where $(v, 1)$ is visually represented with a tip
4. A node side (v, s_v) is spelled as:

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

5. An arc exists 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, this condition holds:

$$\big((v, s_v), (u, s_u)\big) \in A \iff$$

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

This right-hand condition is also known as the (v, u)-oriented-overlap [24].

It's important to note that the notion of node sides allows for treating a k-mer and its reverse complement as the same node entity.

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

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

A path p is considered a "unitig" if its internal nodes have both in-degree and out-degree equal to 1. Additionally, a unitig is said "maximal" if it cannot be extended on either end. To reduce memory usage, a $\mathrm{dBG}(K)$ can be "compacted"

by replacing maximal unitigs with single nodes labeled with the spellings of the unitigs.

An example of a compacted dBG(K), with $k = 4$, is presented in Figure 1. In this example, the maximal unitig $(CGAA, GAAA)$ has been replaced with the node $CGAAA$.

2.2 Fast and Succint k-mer Set Compression

A k-mer set K can be compressed by creating a representation S consisting of strings of arbitrary length, such that the set of all k-mers extracted from S precisely matches the original set K.

The "spectrum" of a set of strings S is defined as the set of all k-mers and their reverse complements that occur in at least one string $s \in S$, formally

$$spec_k(S) = \{t \in \Sigma^k | \exists s \in S : t \text{ or } rc(t) \text{ is substring of } s\}$$

Consider a set of input strings K, each composed of k characters. Our goal is to find a minimal collection of strings that maintains a particular spectrum. Formally, this can be defined as follows:

Definition 1. *A Spectrum Preserving String Set (SPSS) for the input set of k-mers K is a collection of strings, denoted as S, where each string in S has a length of at least k, and such that $spec_k(K) = spec_k(S)$.*

The key property of the SPSS is its ability to contain the same collection of k-mers as the original input set K, including both the k-mers themselves and their reverse complements. Notably, our definition permits the repetition of k-mers and their reverse complements, whether within a single string or spread across multiple strings.

A straightforward method to assess the size of a string set S is by computing its *cumulative length* defined as the sum of all the string lengths:

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

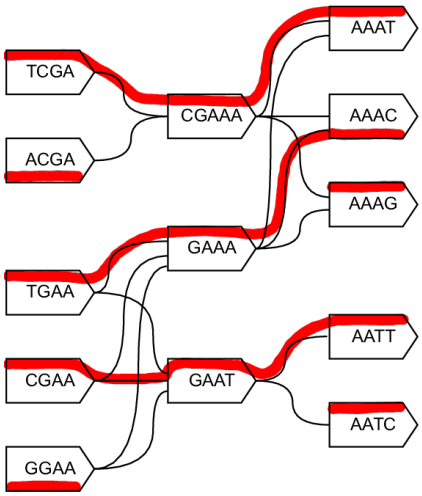
where $|s|$ is the length of the string s.

Problem 1. Given as input a k-mer set K, find the Spectrum Preserving String Set S with minimum $CL(S)$.

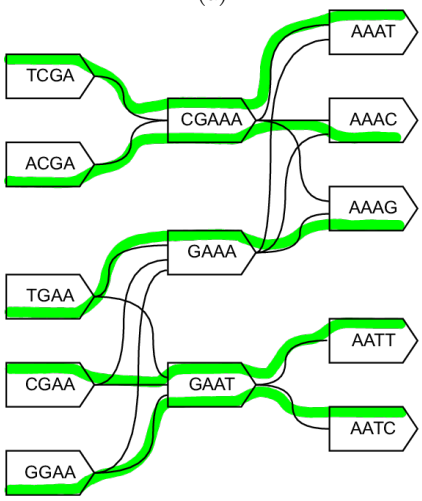
Essentially, our objective is to identify the smallest possible set of strings, each at least k characters long, that preserves the k-mer spectrum of the original input strings. This means that k-mers and their reverse complements may be repeated within or across these strings. This approach differs from simplitigs, which impose an additional constraint on the repetition of k-mers.

It has been shown in [30] that Problem 1 can be solved exactly in polynomial time. The authors introduced an algorithm that utilizes a many-to-many minimum-cost path query combined with a minimum-cost perfect matching

Fig. 1: An example of a compacted de Bruijn graph with $k = 4$ is shown. Nodes are labeled with k-mers. In Figure (a), simplitigs are represented by the red disjoint path cover, where each node is traversed only once, resulting in a total cumulative length (CL) of 35. In Figure (b), an SPSS is represented by the green path cover computed by USTAR2, where nodes can be reused, leading to a total CL of 32. Figure reported also in [29].



(a)



(b)

strategy. However, this optimal algorithm, named Matchtigs, demands $O(n^3m)$ time complexity, where n represents the number of nodes in the de Bruijn graph and m the number of arcs, making it impractical for processing large datasets.

The objective of our study is to develop an effective heuristic capable of handling large datasets efficiently while achieving high-quality compression of k-mers.

Consider again the example in Figure 1. From the path $p = (GGAA, GAAT, AATC)$ we can compute its spell $spell(p) = GGAATC$ that contains all the 4-mers $GGAA$, $GAAT$ and $AATC$ in p, with a saving of 6 bases. Thus from a set of paths P that contains all the nodes in dBG(K) we can derive a set S of strings that represent all the k-mers in K, that is, an SPSS of K. Therefore a path cover can be used in order to compute S and thus compress the k-mer set. Note that the paths can pass through the same node more than once.

In the context of simplitigs, where k-mers are not allowed to be reused, various greedy and non-optimal algorithms have been proposed. ProphAsm [4] employs a straightforward heuristic: it selects an arbitrary k-mer from the de Bruijn graph (dBG(K)), attempts to extend it both forwards and backwards as much as possible, and repeats this process until all k-mers have been covered. Similarly, UST [24] uses the compacted dBG(K) generated by BCALM2. It starts with an arbitrary node, extends it forward as far as possible, and then restarts with available nodes, ultimately merging linked paths. Both methods follow a comparable approach of selecting an initial k-mer and extending it without taking the entire graph structure into account.

For instance, as illustrated in Figure 1, ProphAsm and UST, by choosing nodes arbitrarily, may generate simplitigs similar to those shown in Figure 1 (a). When computing the spelling of each path, we obtain the set of 7 strings $S = \{TCGAAAT, ACGA, TGAAAC, AAAG, CGAATT, GGAA, AATC\}$ with a cumulative length $CL(S) = 35$.

To solve Problem 1 and construct a minimum SPSS using the Matchtigs algorithm, the time complexity will be $O(n^3m)$. As discussed by the authors, this exact algorithm is impractical for large datasets. In the case of simplitigs, it is evident that both ProphAsm and UST are not optimal algorithms. However, their underlying structure is quite efficient, as each node is processed only once, resulting in linear complexity.

In this work, we present USTAR2 (Unitig STitch Advanced constRuction) a linear compression algorithm that exploits the connectivity of the dBG graph, to ensure a good compression ratio, while efficiently traversing the graph.

USTAR2 employs a heuristic to approximate a minimum SPSS, similarly to UST, leveraging the compacted de Bruijn graph generated by BCALM2. Like UST and ProphAsm, USTAR2 begins by choosing a seed node from the graph and attempts to construct a path starting from this node. The path is built by linking adjacent nodes until extension is no longer possible. The process is repeated with new seed nodes until every node is covered by a path. The effectiveness of USTAR2 hinges on two crucial operations: selecting an optimal seed node and extending the path using available connections.

Algorithm 1: USTAR2

```
Data: de Bruijn graph dBG; parameter D
Result: SPSS S
begin
    S = ∅
    seed-nodes = sort nodes by Imb(node)
    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}
    return S
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}
    L = {neighbors of contig head}
    level = 1
    found new node = false;
    while level<=D and not found new node do
        L = {neighbors of all nodes in L}
        level=level+1
        L = Filter(L)
        L′ = {non-visited nodes in L}
        if L′ not empty then
            k = less connected node in L′
            visit(k)
            found new node = true;
            p = path from k to contig head
            contig = merge(p, contig)
    if found new node then
        return Extend(contig)
    else
        return contig
Function Filter(L):
    for v ∈ L do
        p = path from v to contig head
        if length(p) > 2k - 2 then
            remove v from L
    return L
```

The pseudocode of USTAR2 is shown in Algorithm 1.

In order to select a good seed node, we need to ensure that this node will be part of an optimal path for the SPSS problem. We define the imbalance of a node, $Imb(v) = |OutDegree(v) - InDegree(v)|$, as the absolute difference between the out-degree and in-degree of the node v. In general, if a node is balanced, that is $Imb(v) = 0$, it is very likely to be traversed by one or more paths. However, as proved in [30], if a node is imbalanced $Imb(v) \neq 0$, it must be the starting or ending point of some optimal path. Therefore, balanced nodes should be avoided as starting points; instead, only nodes with imbalances should be chosen as seed nodes. To capitalize on this insight, USTAR2 selects the most imbalanced node, i.e. the one with the highest $Imb(v)$ value, as the seed node at each iteration.

The topological properties of the de Bruijn graph are important, not only for the identification of good seed nodes but also for the construction of a path from this node.

In constructing the path cover for simplitigs, we have noted that both UST and ProphAsm may select highly connected nodes [26]. Since these nodes become unavailable in subsequent iterations, such choices can result in isolated nodes, thereby increasing the cumulative length. As illustrated in Figure 1 (a), the path cover for simplitigs creates four isolated nodes, which contribute significantly to the high CL value. This issue persists in the context of SPSS, even though nodes can be visited multiple times.

In USTAR2, we aim to prevent this issue by adopting a different strategy. During each iteration, we extend the current path by selecting the node with fewer connections, ensuring that highly connected nodes remain available for later use. This approach helps reduce the CL by producing fewer, longer strings, which minimizes the likelihood of generating isolated nodes. Just like in UST, USTAR2 extends the current seed node first to the right and then to the left to construct a contig.

When the current contig can no longer be extended due to all neighboring nodes having been visited, we attempt to reuse one of these nodes to connect to an unvisited node. This search is conducted using a breadth-first search (BFS) with a depth limit D to restrict the exploration range. During each iteration, we prune the search space by eliminating branches that would increase the cumulative length (CL). Specifically, we discard nodes that would produce a path longer than $2k - 2$, as extending beyond this length would not improve the CL and would be better served by creating two separate contigs. If multiple unvisited nodes are available, we select the one with fewer connections to maintain consistency with our previous strategy.

In terms of running time, this algorithm will traverse efficiently the de Bruijn graph. In order to efficiently implement the above algorithm, we can take advantage of efficient data structures to store the most important information. For example, the list of seed nodes can be ordered in $O(n)$ time, where n is the number of nodes, using radix sort, since we know the maximum imbalance of a node. Also, the nodes can be ordered by the number of connections so that we can select the less connected node in constant time. Since we are dealing with DNA

sequences, the alphabet is composed of only 4 bases and consequently, each node in the de Bruijn graph can have at most 4 neighbors. Thus, in the breadth-first search at level D we can visit at most 4^D nodes. However, in real cases, the distribution of k-mers will produced a sparse de Bruijn graph. Moreover, the filtering step will reduced the number of visited nodes and thus this value is only an upper bound. Overall, in the traversal of the graph with the function $Extend$ each iteration takes constant time, where the constant is at most 4^D. In summary, the above algorithm runs in linear time on n, the number of nodes in the graph.

In the example in Figure 1 (b), we can see that allowing multiple traversals of a node enables USTAR2 to construct a minimum-sized SPSS. USTAR2 ensures that the most connected node, $GAAA$, is initially avoided while building the first paths. This results in a path cover (shown in green) of the de Bruijn graph, forming a set of five strings: $S' = \{TCGAAAT, ACGAAAC, TGAAAG, CGAATT, GGAATC\}$. The cumulative length of S' is $CL(S') = 32$. Overall we obtain

$$CL(S') = 32 < CL(S) = 35 < CL(K) = 53$$

In this scenario, the uncompressed k-mer set would require a cumulative length of $CL(K) = 53$. With simplitigs, the k-mer set can be compressed to $CL(S) = 35$, whereas the SPSS computed by USTAR2 achieves a more efficient compression with $CL(S') = 32$.

3 Results

In this section, we present the results of a series of experiments designed to determine the most effective tool for compressing k-mers. To evaluate the performance of our tool, USTAR2, we conducted comparative assessments against several existing tools: UST [24], USTAR [26], Matchtigs, and Greedy Matchtigs [30].

To conduct our assessments, we utilized a collection of real read datasets obtained from the NCBI's Sequence Read Archive that was selected by previous studies [21,25,14,7,4]. A summary of the properties of these datasets is provided in Table 1.

For each dataset, we computed all k-mers for different values of k (see Table 7 in the Appendix); the k-mer lengths considered are 15, 21, 31, and 41. All the tools in examination require the preliminary construction of a compacted de Bruijn graph (dBG) and an input file in the format proposed by the authors of BCALM2[7]; for this reason all datasets are preprocessed by BCALM2 once for all. For USTAR2 we use the default parameter $D = 7$.

The following experiments are executed on a server equipped with Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz and 100GB of RAM, bounding the maximum amount of memory available.

In the first experiment, we ran UST, USTAR, USTAR2, Greedy Matchtigs and Matchtigs using the k-mer sets of all datasets with $k = 21$. The results are presented in Table 2 in which are reported the cumulative length (CL) of the

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

Dataset	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

sequences computed with each tool. We were able to run Matchtigs only for the datasets with the smaller number of $k-mers$, since for the other datasets it gave out-of-memory errors, meaning that 100GB of RAM were not sufficient to run the program. This behavior was expected, since also the authors of Matchtigs[30] reported the problem. To bypass this problem, we included in the analysis also the greedy version of Matchtigs that produces very similar results to the optimal counterpart.

As anticipated, Matchtigs' algorithm, being exact, generates the optimal k-mers representation. USTAR2 ranks second, consistently outperforming all other tools and achieving the most compact representations for all datasets, coming close to the optimal.

We can observe that tools based on simplitigs, such as UST and USTAR, are unable to compress k-mers effectively due to the k-mer uniqueness constraint. In contrast, USTAR2 and Greedy Matchtigs, which use the more general SPSS approach allowing k-mer reuse, achieve significantly better results in terms of CL. Among USTAR2's competitors, Greedy Matchtigs stands out as the top performer, particularly in scenarios where Matchtigs cannot be run due to memory limitations.

In evaluating the quality of compression, we utilized two separate metrics: CL (Cumulative Length), as described in Section 2.2, which evaluates quality prior to compressing k-mers; and *compression*, which refers to the size of the file containing the string representation of k-mers after compression using the dedicated compressor MFCompress[22].

In our subsequent experiment, we evaluated the compressibility of different representations by comparing their file sizes after compression. The results are detailed in Table 3. As previously stated, Matchtigs failed to run on several datasets due to out-of-memory errors; furthermore, MFCompress crashed when processing dataset SRR5853087_1. Overall, Greedy Matchtigs emerged as the

Table 2: Considering $k = 21$, the datasets are processed with the following tools: UST, USTAR, USTAR2, Greedy Matchtigs, and Matchtigs. For each tool, the cumulative length (CL) is provided. Note that Matchtigs was only able to process four datasets due to out-of-memory errors with the others. The average CL across all experiments is shown in the final row in which USTAR obtained the best value and it is highlighted in bold.

K=21	**CL**				
	UST	**USTAR**	**USTAR2**	**Greedy Matchtigs**	**Matchtigs**
SRR001665_1	36,357,928	36,324,848	33,638,588	33,858,376	33,380,903
SRR001665_2	45,751,142	45,694,102	41,864,643	42,201,273	41,478,989
SRR061958_1	623,862,618	191,039,506	178,434,526	179,301,460	
SRR061958_2	767,654,838	211,459,109	198,026,097	198,820,674	
SRR062379_1	252,418,995	248,519,235	226,998,129	228,491,551	
SRR062379_2	246,073,774	241,478,754	220,352,087	221,708,614	
SRR10260779_1	188,012,488	184,854,088	170,629,253	171,477,677	
SRR10260779_2	214,245,523	210,202,663	192,382,255	193,409,534	
SRR11458718_1	189,827,141	185,070,581	170,218,475	171,003,884	
SRR11458718_2	202,891,014	196,865,834	179,815,285	180,632,752	
SRR13605073_1	86,006,020	84,974,720	81,822,046	81,970,005	
SRR14005143_1	19,355,339	19,020,479	17,477,215	17,546,167	17,376,011
SRR14005143_2	42,328,593	41,492,693	37,213,243	37,481,708	36,908,792
SRR332538_1	18,649,027	18,382,747	17,615,333	17,688,889	
SRR332538_2	49,648,910	46,689,430	41,053,913	41,226,736	
SRR341725_1	245,548,134	243,816,714	236,221,721	236,557,911	
SRR341725_2	258,344,641	256,477,401	247,741,588	248,138,629	
SRR5853087_1	587,246,289	551,618,109	484,650,727	486,008,368	
SRR957915_1	377,292,074	366,210,794	325,707,476	327,968,686	
SRR957915_2	579,294,390	562,058,930	501,809,029	505,129,713	
average	251,540,444	197,112,537	**180,183,581**	181,031,130	

most effective compressor, with USTAR2 performing nearly as well, while USTAR and UST showed significantly poorer results.

The length of k-mers is a crucial parameter in many bioinformatics applications. In several studies [35,34,17,1], a k-mer length of 31 is used, balancing the need to capture sequence context with computational efficiency. Therefore, we investigated the behavior of CL and *compression* across different k-mer lengths ($k \in \{15, 21, 31, 41\}$). The average results for CL and *compression* are presented in Tables 4 and 5, respectively. Matchtigs was excluded from these experiments as it could only be run on a limited subset of the datasets.

USTAR2 consistently showed superior CL across various k-mer sizes, outperforming other tools except for $k = 15$. Notably, it achieved the best compression results for all k-mer lengths except $k = 15$. In contrast, UST and USTAR consistently demonstrated significantly lower compression performance, with UST producting an average 44% larger files compared to USTAR2. This comparative analysis underscores the dominance of USTAR2 and Greedy Matchtigs over

Table 3: File size of k-mers (with $k = 21$) after compression using MFCompress. Dataset SRR5853087_1 encountered a compression error. Several Matchtigs computations failed previously due to out-of-memory errors.

k=21	**Compression**				
	UST	**USTAR**	**USTAR2**	**Greedy Matchtigs**	**Matchtigs**
SRR001665_1	12,641,658	12,332,551	8,700,726	8,813,736	8,845,254
SRR001665_2	15,492,263	15,109,673	10,860,629	11,003,600	10,876,474
SRR061958_1	194,173,905	185,905,825	45,319,763	45,454,536	
SRR061958_2	235,657,588	225,975,765	50,305,926	50,486,848	
SRR062379_1	82,713,766	79,283,723	58,666,732	58,566,163	
SRR062379_2	80,164,746	76,708,406	56,647,674	56,882,630	
SRR10260779_1	64,644,700	61,724,139	43,092,171	43,311,649	
SRR10260779_2	72,772,294	69,375,320	48,707,100	48,574,622	
SRR11458718_1	64,694,925	61,236,404	42,574,564	42,645,409	
SRR11458718_2	68,982,466	65,438,050	44,755,366	44,708,191	
SRR13605073_1	25,833,347	24,546,244	20,112,062	20,144,454	
SRR14005143_1	6,419,520	6,220,215	4,208,321	4,179,654	4,194,213
SRR14005143_2	13,117,896	12,655,430	8,983,371	8,932,076	8,980,170
SRR332538_1	5,737,778	5,599,034	4,378,514	4,393,504	
SRR332538_2	14,410,775	13,528,977	9,813,628	9,712,821	
SRR341725_1	80,436,678	78,193,253	60,734,955	61,160,751	
SRR341725_2	84,250,689	81,877,574	63,841,063	64,009,811	
SRR5853087_1					
SRR957915_1	122,748,678	116,872,195	83,085,402	82,631,218	
SRR957915_2	182,073,051	172,757,385	129,433,290	130,303,345	
average	75,103,512	71,860,009	**41,801,119**	41,890,264	

Table 4: Average Cumulative Length (CL) across datasets for different values of $k \in \{15, 21, 31, 41\}$ among UST, USTAR, USTAR2, and Greedy Matchtigs. Except for $k = 15$, USTAR2 achieved the highest CL (highlighted in bold), followed by Greedy Matchtigs, USTAR, and UST.

Cumulative Length				
k	**UST**	**USTAR**	**USTAR2**	**Greedy Matchtigs**
15	389,526,197	225,048,269	122,088,205	**118,257,100**
21	251,540,444	197,112,537	**180,183,581**	181,031,130
31	296,999,909	293,947,184	**217,688,497**	218,851,662
41	366,686,084	364,249,733	**269,841,393**	271,412,813

UST and USTAR in terms of both CL and compression metrics. Specifically, at $k = 31$, the most commonly used k-mer size, USTAR2 exhibited the highest CL and *compression* performances among all evaluated tools.

Table 5: Average *compression*, measured as the size reduction of the fasta file using MFCompress, across all datasets for different values of $k \in \{15, 21, 31, 41\}$ among UST, USTAR, USTAR2, and Greedy Matchtigs.

Compression				
k	**UST**	**USTAR**	**USTAR2**	**Greedy Matchtigs**
15	114,627,914	69,954,835	30,659,296	**29,313,703**
21	75,103,512	71,860,009	**41,801,119**	41,890,264
31	68,297,756	68,585,282	**49,427,296**	49,672,063
41	70,445,678	71,874,051	**50,595,570**	51,137,503

3.1 Time and memory usage

In the previous experiments, we identified USTAR2 and Greedy Matchtigs as the best performing compression algorithms. However, it's important to note that Greedy Matchtigs is based on the exact algorithm of Matchtigs, which may pose efficiency challenges from a computational point of view.

To study the computational performances of these tools, we conducted a comprehensive analysis of their time and memory usage. Currently, USTAR2 runs in a single-threaded mode, whereas Greedy Matchtigs leverages multi-threading capabilities. Initially, we compared these tools under their respective single-threaded configurations.

In the previous tests, we also collected the time and memory consumption across all datasets while varying the k-mer size. The average computational times are depicted in Figure 2, and the corresponding memory requirements are illustrated in Figure 3.

Fig. 2: Average time in seconds (single-thread) used by USTAR2 and Greedy Matchtigs for different k-mer lengths, in a logarithmic scale. USTAR2 execution time is consistently much lower compared to Greedy Matchtigs, resulting in a 96× maximum speedup. Figure reported also in [29].

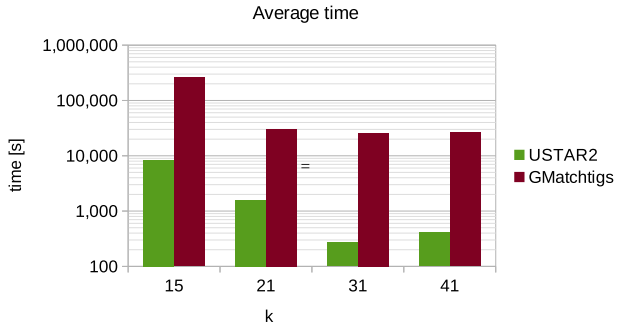


Fig. 3: Average memory in megabyte used by USTAR2 and Greedy Matchtigs for different k-mer lengths. USTAR2 memory usage is generally lower compared to Greedy Matchtigs; in particular, it reaches a peak for $k = 15$, where it consumes half the memory of the competitor. Figure reported also in [29].

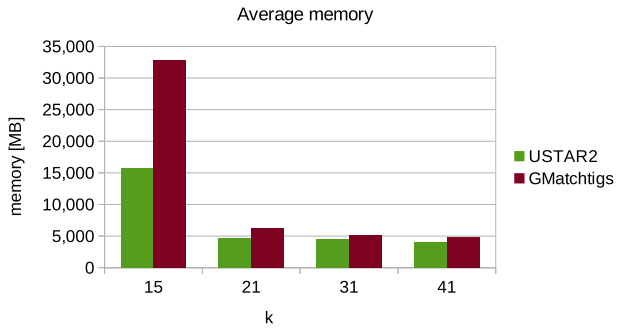
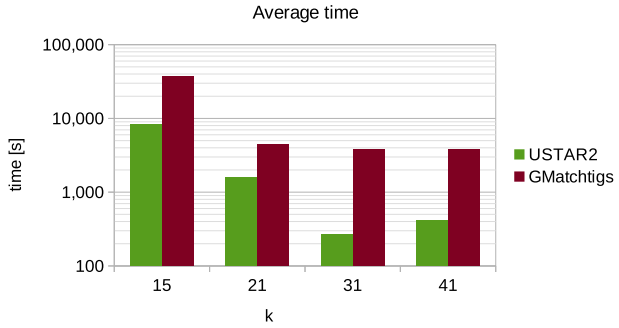


Fig. 4: Average time in seconds used by USTAR2 (single-thread) and Greedy Matchtigs (multi-thread) with different k in logarithmic scale. Even with the advantage of 16 threads, Greedy Matchtigs remains slower than its competitor. In particular, for $k = 31$, USTAR2 gains a 15× speedup. Figure reported also in [29].



From Figures 2 and 3 we can note that there is a notable peak in both time and memory usage when $k = 15$ due to dense de Bruijn graphs at smaller k values. However, USTAR2 shows a decreasing trend in computing time as k increases, whereas Greedy Matchtigs exhibits a plateau after $k = 21$. Conversely, memory requirements decrease with larger values of k for both tools. When comparing the efficiency of the two tools, USTAR2 emerges as more resource-efficient than Greedy Matchtigs. Specifically, at $k = 15$, USTAR2 runs 30 times faster than Greedy Matchtigs while using less than half the memory. At the

commonly used $k = 31$, USTAR2 achieves a remarkable speedup of 96 times compared to Greedy Matchtigs.

Given that Greedy Matchtigs supports multi-threading while USTAR2 does not, we investigated whether parallelization could improve the time performance of Greedy Matchtigs. This comparison is detailed in Figure 4. Running Greedy Matchtigs with 16 threads reduces its computation time compared to its single-threaded version. However, even with 16 threads, Greedy Matchtigs remains significantly slower than USTAR2. For the widely adopted $k = 31$, USTAR2 (single-threaded) is still 15 times faster than Greedy Matchtigs (16-threaded).

In conclusion, USTAR2 not only delivers the best compression performance but also excels in resource efficiency. Moreover, the current single-threaded implementation of USTAR2 has the potential to be further optimized with parallelization techniques.

3.2 Varying paramenter D: Changing BFS depth

In this section we evaluate the behavior of USTAR2 varying the parameter D. Recall that, given a de Bruijn graph, the USTAR2 algorithm begins by selecting a seed and then generates a path by choosing adjacent, unvisited nodes with fewer connections. When no such nodes are available, the algorithm switches to a Breadth First Search (BFS) until it finds an unvisited node or reaches a depth threshold D. The search also stops if the contig formed from the visited nodes exceeds a length of $2k-2$, at which point the same CL is achieved as if a new seed were selected. Here, we study the effect of the parameter D by testing various values and analyzing the corresponding performance. USTAR2 has been executed for different values of D and the same k-mer length $k = 21$ (Tables 8, 9, 10 in the Appendix contains the complete results). A summary of the results is reported in Table 6.

Table 6: Average results obtained by USTAR2 changing the depth parameter D and considering $k = 21$. The best *compression* and *string count* is achieved with $D = 10$ at the cost of processing time. The minimum CL is obtained with $D = 7$.

k=21	USTAR2 -D3	USTAR2 -D5	USTAR2 -D7	USTAR2 -D10	GMatchtigs
CL	182,168,049	180,481,558	180,183,581	180,218,872	181,031,130
SC	2,638,074	2,427,589	2,339,810	2,278,069	2,217,643
compression [bytes]	42,817,578	42,115,904	41,801,119	41,654,047	41,890,264
time [s]	80	843	1,583	3,580	30,265
memory [MB]	4,535	4,601	4,699	4,534	6,062

A side effect of the BFS is a reduction in the number of sequences (also known as *sequence count*, SC) in the SPSS. When the BFS successfully finds a suitable node, it extends the current path rather than starting a new one, thus preventing an increase in SC. This relationship is illustrated in Appendix Table

9, and in Table 6 highlighting the decreasing trend of SC as the paramenter D (BFS depth) increases. USTAR2 has been executed with different values of D and the results confirmed the previously mentioned effect; indeed the highest value of D achieved the lower SC on average.

The same experiment has been done monitoring the *compression* of the file. Even in this case, the best value found is $D = 10$, showing a possible relation between SC and *compression*.

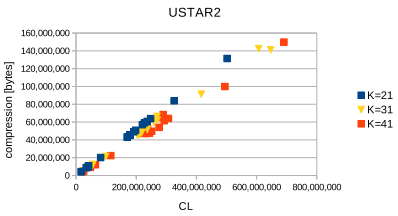
It is important to note that a high value of D is not necessarily optimal. The purpose of the BFS in USTAR2 is to locate an isolated node or a node with neighbors on only one side, which helps extend the contig effectively. However, if the BFS identifies a node with neighbors on both sides, it can prevent the optimization of the CL, as this situation doesn't allow for the best possible extension of the path. Furthermore, increasing D will necessarily increase the execution time (see the summary Table 6). Thus, selecting an appropriate D is crucial to balancing the search depth and minimizing the CL. Accepting a tradeoff between compression ratio and execution time, we selected $D = 7$ as the best depth parameter for USTAR2.

3.3 The relationship of CL, SC and compression

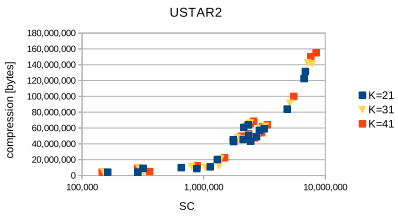
The primary goal of the BFS in USTAR2 is to solve Problem (1) and to minimize the CL by leveraging existing links between sequences, even if this results in repeating k-mers. We aim to demonstrate that a smaller CL significantly benefits final compression, despite the associated SPSS being redundant and containing repeated strings.

Fig. 5: Correlation between *compression* and CL and between *compression* and SC. In Figure (a) the *compression* has a clear linear relation with the CL for all values of k. Figure (b) shows an increasing trend with SC, in particular, the slope of the curve emphasizes the difficulty of compressing a high number of sequences.

(a) Cumulative length.



(b) String count.



As observed in Tables 4 and 5, tools that achieve a smaller CL also attain superior compression results. This correlation is further highlighted in Figure 5a,

which reveals a clear linear relationship between CL and compression for various k-mer lengths. This linear relationship underscores the critical importance of minimizing CL to enhance compression efficiency.

Moreover, we explore the relationship between the string count (SC) and final compression. Figure 5b illustrates that reducing SC has a more significant impact when SC is high, as indicated by the steeper slope of the curve. This suggests that optimizing SC can substantially improve compression, particularly in scenarios with a high string count.

The correlations presented in Figures 5 indicate that both CL and SC play crucial roles in optimizing sequence file compression. Since optimal compression is achieved with both low CL and low SC, it is essential to minimize both metrics. Currently, all tools are based on Problem (1) and they only focus on minimizing CL. However, to further enhance compression performance, additional research is needed to develop strategies for effectively minimizing SC as well.

4 Conclusions

In this paper, we present USTAR2, an advanced software tool developed for the compression of k-mer sets. Our method tackles the path cover problem on a de Bruijn graph to find an optimal representation of the k-mer set, focusing on minimizing the cumulative length of the compressed data. By strategically making decisions based on node connectivity and reusing previously traversed nodes, USTAR2 achieves compression ratios that significantly outperform established tools like UST and USTAR that do not allow repeated k-mer. Additionally, USTAR2 demonstrates greater efficiency in k-mer set representation compared to Greedy Matchtigs, excelling in both cumulative length and compression across most k values.

We conducted a comprehensive performance evaluation of USTAR2 using various datasets and performed a comparative analysis with alternative tools, including Matchtigs and Greedy Matchtigs. The results consistently demonstrate the superiority of our approach, with USTAR2 outpacing Greedy Matchtigs in terms of both time and memory utilization. While USTAR2 and Greedy Matchtigs achieve similar compression ratios, USTAR2 exhibits a remarkable speed advantage, being 52 times faster on average and requiring less memory, particularly for lower k-mer lengths. For the commonly used value of $k = 31$, this speed advantage rises to 96 times. Even when Greedy Matchtigs utilizes multithreading, USTAR2 maintains a significant performance advantage, remaining 15 times faster when executed with a single thread.

The BFS phase is a crucial step in the algorithm that permits an effective reuse of k-mers and a reduction in the string count. By selecting the right depth D, USTAR2 achieves a good compromise between compression and execution time that can be adapted to enhance compression. The current single-threaded operation suggests potential for improved performance through parallelization,

which needs further investigation. Thus USTAR2 offers an effective and resource-efficient solution for compressing k-mer sets.

Lastly, the clear correlations of cumulative length and sequence count with final compression highlight the importance of minimizing the number of sequences in addition to cumulative length. Currently, the problem definition and the proposed solution do not focus on the reduction of SC but only of CL. We proved that, not only a string data structure with an index can greatly benefit from a low string count as stated by [30], but even the final compression. Therefore developing strategies that minimize both factors at the same time could further enhance compression ratios for many bioinformatics pipelines based on k-mers.

ACKNOWLEDGEMENTS

Authors are supported by the Project funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.4 - Call for tender No. 3138 of 16 December 2021, rectified by Decree n.3175 of 18 December 2021 of Italian Ministry of University and Research funded by the European Union – NextGenerationEU.

References

1. Andreace, F., Pizzi, C., 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). https://doi.org/10.1089/cmb.2021.0270
2. 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)
3. Bradley, P., Den Bakker, H.C., Rocha, E.P., McVean, G., Iqbal, Z.: Ultrafast search of all deposited bacterial and viral genomic data. Nature biotechnology **37**(2), 152–159 (2019)
4. Břinda, K., Baym, M., Kucherov, G.: Simplitigs as an efficient and scalable representation of de bruijn graphs. Genome biology **22**(1), 1–24 (2021)
5. Cavattoni, M., Comin, M.: Classgraph: Improving metagenomic read classification with overlap graphs. Journal of Computational Biology **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 Computing Surveys (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 (01 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. Dufresne, Y., Lemane, T., Marijon, P., Peterlongo, P., Rahman, A., Kokot, M., Medvedev, P., Deorowicz, S., Chikhi, R.: The K-mer File Format: a standardized and compact disk representation of sets of k-mers. Bioinformatics **38**(18), 4423–4425 (07 2022)

11. Harris, R.S., Medvedev, P.: Improved representation of sequence bloom trees. Bioinformatics **36**(3), 721–727 (2020)
12. Holley, G., Melsted, P.: Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs. Genome biology **21**, 249 (09 2020). https://doi.org/10.1186/s13059-020-02135-8
13. Khan, J., Kokot, M., Deorowicz, S., Patro, R.: Scalable, ultra-fast, and low-memory construction of compacted de bruijn graphs with cuttlefish 2. Genome Biology **23**, 190 (2022)
14. Kokot, M., Długosz, M., Deorowicz, S.: Kmc 3: counting and manipulating k-mer statistics. Bioinformatics **33**(17), 2759–2761 (2017)
15. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics **25**(14), 1754–1760 (05 2009)
16. 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)
17. 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, February 9-11, 2022. pp. 62–70 (2022)
18. Monsu, M., Comin, M.: Fast alignment of reads to a variation graph with application to snp detection. Journal of Integrative Bioinformatics **18**(4), 20210032 (2021)
19. Ondov, B.D., Treangen, T.J., Melsted, P., Mallonee, A.B., Bergman, N.H., Koren, S., Phillippy, A.M.: Mash: fast genome and metagenome distance estimation using minhash. Genome biology **17**(1), 1–14 (2016)
20. 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 systems **7**(2), 201–207 (2018)
21. Pandey, P., Bender, M.A., Johnson, R., Patro, R.: Squeakr: an exact and approximate k-mer counting system. Bioinformatics **34**(4), 568–575 (2018)
22. Pinho, A.J., Pratas, D.: Mfcompress: a compression tool for fasta and multi-fasta data. Bioinformatics **30**(1), 117–118 (2014)
23. Qian, J., Comin, M.: Metacon: Unsupervised clustering of metagenomic contigs with probabilistic k-mers statistics and coverage. BMC Bioinformatics **20**(367) (2019). https://doi.org/10.1186/s12859-019-2904-4
24. Rahman, A., Medvedev, P.: Representation of k-mer sets using spectrum-preserving string sets. In: International Conference on Research in Computational Molecular Biology. pp. 152–168. Springer (2020)
25. Rizk, G., Lavenier, D., Chikhi, R.: Dsk: k-mer counting with very low memory usage. Bioinformatics **29**(5), 652–653 (2013)
26. Rossignolo, E., Comin, M.: Ustar: Improved compression of k-mer sets with counters using de bruijn graphs. In: Guo, X., Mangul, S., Patterson, M., Zelikovsky, A. (eds.) Bioinformatics Research and Applications. pp. 202–213. Springer Nature Singapore, Singapore (2023)
27. Rossignolo, E., Comin, M.: Enhanced compression of k-mer sets with counters via de bruijn graphs. Journal of Computational Biology **31**(6), 524–538 (2024). https://doi.org/10.1089/cmb.2024.0530
28. Rossignolo, E., Comin, M.: Ustar2: Fast and succinct representation of k-mer sets using de bruijn graphs. In: Proceedings of the 17th International Joint Conference

on Biomedical Engineering Systems and Technologies - Volume 1: BIOINFORMATICS. pp. 368–378. INSTICC, SciTePress (2024). https://doi.org/10.5220/0012423100003657
29. Rossignolo, E., Comin, M.: Ustar2: Fast and succinct representation of k-mer sets using de bruijn graphs. In: BIOSTEC (1). pp. 368–378 (2024)
30. Schmidt, S., Khan, S., Alanko, J.N., Pibiri, G.E., Tomescu, A.I.: Matchtigs: Minimum plain text representation of k-mer sets. Genome Biology (Online) **24** (2023). https://doi.org/10.1186/s13059-023-02968-z
31. Schmidt, S., Alanko, J.N.: Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time. Research square p. rs.3.rs—2581995 (February 2023). https://doi.org/10.21203/rs.3.rs-2581995/v1
32. Storato, D., 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). https://doi.org/10.1109/TCBB.2021.3117406
33. Sun, C., Harris, R.S., Chikhi, R., Medvedev, P.: Allsome sequence bloom trees. Journal of Computational Biology **25**(5), 467–479 (2018)
34. Sun, C., Medvedev, P.: Toward fast and accurate snp genotyping from whole genome sequencing data for bedside diagnostics. Bioinformatics **35**(3), 415–420 (2019)
35. Wood, D.E., Salzberg, S.L.: Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome biology **15**(3), 1–12 (2014)

APPENDIX

Table 7: Number of k-mers for each dataset varying $k \in \{15, 17, 21, 31, 41\}$.

dataset	**#15-mers**	**#21-mers**	**#31-mers**	**#41-mers**
SRR001665_1	13,889,837	14,286,068	10,343,472	-
SRR001665_2	16,371,558	16,895,362	12,058,109	-
SRR061958_1	225,788,025	388,490,798	404,149,685	392,492,657
SRR061958_2	265,935,616	482,235,278	495,804,915	475,405,235
SRR062379_1	109,810,585	152,875,155	160,692,477	160,746,342
SRR062379_2	108,958,432	151,987,994	159,905,793	158,802,318
SRR10260779_1	84,250,397	113,667,728	123,624,245	127,090,699
SRR10260779_2	93,032,179	128,074,943	139,633,894	143,150,103
SRR11458718_1	89,998,269	126,431,861	137,995,280	143,397,012
SRR11458718_2	94,018,791	134,997,414	150,549,990	159,144,668
SRR13605073_1	43,488,336	54,085,000	55,764,573	54,682,553
SRR14005143_1	11,307,338	13,223,059	15,005,192	16,272,583
SRR14005143_2	23,691,810	28,456,533	31,850,681	33,872,511
SRR332538_1	10,624,064	11,404,027	11,382,816	10,666,430
SRR332538_2	18,741,106	25,674,930	28,880,136	27,477,871
SRR341725_1	132,442,790	188,913,254	185,618,107	176,391,089
SRR341725_2	136,484,353	196,035,961	192,133,588	181,970,438
SRR5853087_1	159,744,051	316,438,109	382,773,071	399,026,650
SRR957915_1	126,236,121	208,110,514	239,200,400	250,988,377
SRR957915_2	188,867,779	335,926,750	364,597,018	361,352,380

Table 8: Variations on the Cumulative Length (CL) considering different depths D for the BFS. On average, the best depth found is 7 for $k = 21$.

k=21	**CL**				
	USTAR2 -D3	**USTAR2 -D5**	**USTAR2 -D7**	**USTAR2 -D10**	**GMatchtigs**
SRR001665_1	33,724,442	33,614,577	33,638,588	33,652,878	33,858,376
SRR001665_2	42,032,844	41,820,857	41,864,643	41,899,248	42,201,273
SRR061958_1	179,344,739	178,475,353	178,434,526	178,487,472	179,301,460
SRR061958_2	199,043,149	198,089,216	198,026,097	198,076,635	198,820,674
SRR062379_1	229,141,863	227,227,918	226,998,129	227,098,559	228,491,551
SRR062379_2	222,577,626	220,639,776	220,352,087	220,410,452	221,708,614
SRR10260779_1	172,152,568	170,834,391	170,629,253	170,647,128	171,477,677
SRR10260779_2	194,447,448	192,687,883	192,382,255	192,380,954	193,409,534
SRR11458718_1	171,631,152	170,388,945	170,218,475	170,274,600	171,003,884
SRR11458718_2	181,438,971	179,994,380	179,815,285	179,918,803	180,632,752
SRR13605073_1	82,100,544	81,868,639	81,822,046	81,829,909	81,970,005
SRR14005143_1	17,533,686	17,474,598	17,477,215	17,481,274	17,546,167
SRR14005143_2	37,495,157	37,210,386	37,213,243	37,231,963	37,481,708
SRR332538_1	17,727,239	17,614,331	17,615,333	17,649,124	17,688,889
SRR332538_2	42,042,031	41,234,073	41,053,913	41,098,324	41,226,736
SRR341725_1	236,443,128	236,226,834	236,221,721	236,225,336	236,557,911
SRR341725_2	248,023,531	247,749,586	247,741,588	247,744,041	248,138,629
SRR5853087_1	494,958,774	486,601,863	484,650,727	484,842,177	486,008,368
SRR957915_1	331,279,876	326,641,472	325,707,476	325,639,350	327,968,686
SRR957915_2	510,222,204	503,236,086	501,809,029	501,789,219	505,129,713
average	182,168,049	180,481,558	180,183,581	180,218,872	181,031,130

Table 9: USTAR2 execution with different values of search depth D. The number of sequences (SC) is reported. As expected, the SC decreases with D, showing $D = 10$ as the best value when we optimize the SC.

k=21	**SC**				
	USTAR2 -D3	**USTAR2 -D5**	**USTAR2 -D7**	**USTAR2 -D10**	**GMatchtigs**
SRR001665_1	916,105	885,924	879,448	877,672	872,755
SRR001665_2	1,193,991	1,146,533	1,133,937	1,130,235	1,123,195
SRR061958_1	2,306,853	2,161,535	2,117,504	2,099,684	2,072,990
SRR061958_2	2,545,347	2,394,518	2,347,407	2,327,499	2,299,075
SRR062379_1	3,504,235	3,247,734	3,150,431	3,092,370	3,035,170
SRR062379_2	3,229,276	2,976,275	2,876,695	2,812,434	2,747,027
SRR10260779_1	2,678,185	2,503,565	2,439,910	2,400,804	2,358,361
SRR10260779_2	3,032,485	2,809,268	2,724,523	2,671,897	2,617,288
SRR11458718_1	2,000,191	1,828,965	1,765,388	1,725,923	1,688,024
SRR11458718_2	2,030,450	1,828,201	1,750,189	1,700,710	1,660,098
SRR13605073_1	1,335,525	1,307,461	1,295,437	1,286,375	1,278,422
SRR14005143_1	178,560	165,370	162,631	161,845	159,507
SRR14005143_2	382,346	331,453	317,210	312,625	304,930
SRR332538_1	305,365	292,844	288,003	285,088	283,401
SRR332538_2	773,969	696,268	654,081	616,584	579,922
SRR341725_1	2,196,104	2,150,288	2,141,741	2,138,635	2,131,142
SRR341725_2	2,392,955	2,337,448	2,326,873	2,322,642	2,313,111
SRR5853087_1	8,118,590	7,174,067	6,690,760	6,259,528	5,829,696
SRR957915_1	5,630,951	5,094,097	4,865,589	4,711,944	4,575,254
SRR957915_2	8,009,999	7,219,964	6,868,448	6,626,887	6,423,501
average	2,638,074	2,427,589	2,339,810	2,278,069	2,217,643

Table 10: Size of the compressed sequences in bytes. The file is compressed with the dedicated compressor MFCompress. On average, the smallest size is obtained with the higher search depth $D = 10$, highlighted in bold. Dataset SRR5853087_1 crashed the compressor.

k=21	**compression**				
	USTAR2 -D3	**USTAR2 -D5**	**USTAR2 -D7**	**USTAR2 -D10**	**GMatchtigs**
SRR001665_1	8,870,191	8,728,852	8,700,726	8,694,128	8,813,736
SRR001665_2	11,137,855	10,915,321	10,860,629	10,845,768	11,003,600
SRR061958_1	46,185,168	45,510,962	45,319,763	45,249,118	45,454,536
SRR061958_2	51,493,232	50,801,622	50,305,926	50,229,209	50,486,848
SRR062379_1	60,234,098	59,070,721	58,666,732	58,440,424	58,566,163
SRR062379_2	58,171,336	57,036,189	56,647,674	56,409,550	56,882,630
SRR10260779_1	44,200,753	43,373,952	43,092,171	42,930,594	43,311,649
SRR10260779_2	50,121,123	49,077,343	48,707,100	48,489,648	48,574,622
SRR11458718_1	43,609,615	42,840,309	42,574,564	42,422,045	42,645,409
SRR11458718_2	45,993,597	45,077,154	44,755,366	44,565,472	44,708,191
SRR13605073_1	20,254,872	20,149,898	20,112,062	20,092,184	20,144,454
SRR14005143_1	4,294,726	4,222,948	4,208,321	4,204,745	4,179,654
SRR14005143_2	9,327,582	9,056,375	8,983,371	8,960,433	8,932,076
SRR332538_1	4,438,589	4,393,161	4,378,514	4,371,049	4,393,504
SRR332538_2	10,210,659	9,930,431	9,813,628	9,737,762	9,712,821
SRR341725_1	60,952,863	60,766,288	60,734,955	60,723,741	61,160,751
SRR341725_2	64,106,507	63,879,557	63,841,063	63,826,940	64,009,811
SRR5853087_1	Error				
SRR957915_1	86,252,130	83,947,935	83,085,402	82,520,075	82,631,218
SRR957915_2	133,679,087	131,423,152	129,433,290	128,714,001	130,303,345
average	42,817,578	42,115,904	41,801,119	**41,654,047**	41,890,264