USTAR-CR: Efficient and Compact Compression of k-mer Sets Through Colored de Bruijn Graphs

Enrico Rossignolo,[1] Matteo Comin[1]

[1]Department of Information Engineering, University of Padua, Italy,

{enrico.rossignolo,matteo.comin}@unipd.it

May 29, 2025

Keywords: k-mer Sets; Compression; Colored Bruijn graphs; Plain Text Representation

Abstract:

A core task in computational genomics is transforming input sequences into their constituent k-mers. Efficiently storing these k-mer collections is crucial for scaling bioinformatics workflows. A common strategy involves representing the k-mers as a de Bruijn graph and deriving a compact plain text form through a minimum path cover. In this article, we introduce USTAR-CR, a fast and space-efficient algorithm for compressing multiple k-mer sets. USTAR-CR exploits the structural properties of colored de Bruijn graphs to construct a succinct plain text representation, while also incorporating an effective scheme for encoding k-mer color information. We evaluate USTAR-CR on real sequencing datasets and benchmark it against the state-of-the-art tool GGCAT. USTAR-CR achieves superior compression ratios, significantly reduces memory usage, and offers substantial speed improvements—up to 51× faster—highlighting its effectiveness for large-scale genomic data processing.

https://github.com/CominLab/USTAR-CR

1 Introduction

k-mer-based algorithms have become central tools in bioinformatics, offering scalable and efficient alternatives to traditional sequence alignment approaches. By operating directly on sets of k-mer substrings, these methods bypass the need for full read alignment and enable fast, memory-efficient analysis pipelines. Over the past decade, they have seen widespread adoption due to their conceptual simplicity and ability to handle large-scale data. These approaches have demonstrated exceptional performance across a range of applications. In genome assembly, tools such as SPAdes (Bankevich et al., 2012) leverage k-mer-based strategies to reconstruct complete genomes with high accuracy. In metagenomics, a variety of tools (Wood and Salzberg, 2014; Andreace et al., 2021; Qian and Comin, 2019; Cavattoni and Comin, 2023; Storato and Comin, 2022) employ k-mers to classify microbial content in complex samples, achieving speedups of up to 900× compared to traditional methods like MegaBLAST. Similarly, in genotyping, k-mer-centric tools (Denti et al., 2019; Sun and Medvedev, 2019; Marcolin et al., 2022; Monsu and Comin, 2021) identify genetic variants across individuals and populations without reliance on full-sequence alignment. In phylogenomics, Mash (Ondov et al., 2016) uses k-mers to estimate genomic distances, facilitating fast evolutionary analysis. Numerous other tools (Harris and Medvedev, 2020; Marchet et al., 2020) also apply k-mer techniques to enable rapid sequence search over large genomic databases.

The scalability of k-mer methods lies in their ability to handle datasets containing billions of k-mers. A key factor in their performance is how these sets are represented. Depending on the application, this choice balances between time-efficient representations with good cache locality, and compressed formats that reduce memory footprint. As sequencing data grows, minimizing the space required to store and query k-mer sets has become an active area of research. Conway and Bromage (Conway and Bromage, 2011) showed that storing n k-mers requires at least $\log \binom{4^k}{n}$ bits in the worst case. However, real-world datasets typically contain redundant or overlapping k-mers, enabling more compact representations (Chikhi et al., 2021).

A widely adopted method to reduce redundancy is to organize the k-mer set K into maximal unitigs, derived from the de Bruijn graph. In this graph, nodes represent k-mers and edges indicate $(k-1)$-length overlaps. A unitig corresponds to a non-branching path in this graph. Each unitig u is encoded as a string spell(u) of length $|u| + k - 1$, where $|u|$ is the number of k-mers it contains. For example, the unitig (AAC, ACG, CGT) is represented as the string $AACGT$. This approach reduces space by overlapping the shared regions between adjacent k-mers. The full k-mer set K can thus be compactly encoded as a set of such unitigs U, where every k-mer in K appears as a substring of some spell(u), $u \in U$.

An important extension of the de Bruijn graph is the colored de Bruijn graph, which allows for the representation of multiple datasets simultaneously. Initially proposed for applications like de novo assembly and genotyping (Andreace et al., 2023; Iqbal et al., 2012), this structure annotates each k-mer with the dataset identifiers (colors) in which it appears. Colored de Bruijn graphs enable compact joint representations while retaining dataset-specific provenance.

This model is widely used in pangenomics (Zekic et al., 2018), RNA-seq quantification (Bray et al., 2016), microbial classification (Luhmann et al., 2021), and related domains.

The current state-of-the-art for colored k-mer compression is GGCAT (Cracco and Tomescu, 2023), which builds compacted colored de Bruijn graphs by combining k-mer counting and unitig generation with efficient color encoding. GGCAT outperforms earlier tools like Cuttlefish 2 (Khan et al., 2021) and BiFrost (Luhmann et al., 2021), offering significant improvements in both compression and query speed. However, its computational and memory demands remain high. For instance, compressing 649K bacterial genomes with GGCAT requires more than 21 GB of RAM and over 11 hours of runtime on a high-performance machine.

In this work, we introduce USTAR-CR[1] (Unitig STitch Advanced constRuction with Colors Reordering), a fast and memory-efficient algorithm for compressing multiple k-mer sets using a plain text representation of colored de Bruijn graphs. USTAR-CR generates a compact spectrum-preserving string set (SPSS) while supporting efficient color storage, enabling scalable compression of large genomic datasets with minimal computational overhead.

1.1 Related Works

Plain text representations of k-mer sets have become a widely used strategy for practical and efficient data compression. Formally, such representations are defined as spectrum-preserving string sets (SPSS)—collections of strings that include all k-mers from the input data (including reverse complements), while excluding extraneous k-mers. This concept ensures that the original k-mer spectrum is preserved without redundancy.

[1]A preliminary version of USTAR-CR has been presented at ICCABS 2025 (Rossignolo and Comin, 2025b).

Initially, Rahman and Medvedev (Rahman and Medvedev, 2020) and Břinda, Baym, and Kucherov (Břinda et al., 2021) independently proposed methods for constructing such representations without repeated k-mers. Rahman and Medvedev introduced the SPSS framework, while Břinda et al. coined the term simplitigs. To avoid ambiguity with the later, broader definition of SPSS (which allows repeated k-mers), we adopt simplitigs to specifically refer to representations where each k-mer appears only once.

Both UST (from Rahman and Medvedev) and ProphAsm (from Břinda et al.) apply greedy heuristics to merge k-mers into longer strings. UST builds a node-centric de Bruijn graph to extend unitigs, whereas ProphAsm uses a hash-based extension strategy without explicit graph construction. These techniques aim to reduce two key metrics: the cumulative length (CL)—the total number of characters in the representation—and the string count (SC)—the number of separate strings. Lowering CL decreases the memory needed to store the strings, while reducing SC simplifies indexing, leading to overall storage savings.

Building on these ideas, the USTAR heuristic (Rossignolo and Comin, 2023, 2024a) was recently introduced. USTAR improves the traversal of de Bruijn graphs by leveraging graph connectivity and local density to guide the construction of longer and fewer paths. This connectivity-aware strategy often leads to better compression than earlier greedy approaches, particularly on dense graphs.

An important breakthrough came with Matchtigs (Schmidt et al., 2023), which introduced the first algorithm to compute an SPSS of minimum cumulative length, while allowing repeated k-mers. Matchtigs formulates the problem as a min-cost path cover with a many-to-many path matching strategy, solvable in polynomial time. However, its high computational complexity—$O(n^3m)$ for n nodes and m arcs—makes it impractical for large graphs. To address this, the authors also proposed a faster, heuristic version called Greedy Matchtigs, which trades optimality for efficiency and produces comparably compact representations at a fraction of the cost. The recent USTAR2 algorithm (Rossignolo and Comin, 2024b, 2025a) further advances SPSS construction by introducing an efficient path cover heuristic that strategically reuses previously visited nodes. USTAR2 achieves compression ratios similar to Greedy Matchtigs, with significantly better runtime and memory usage.

While these methods focus on compressing single k-mer sets, GGCAT remains the leading tool for handling multiple datasets through compacted colored de Bruijn graphs. GGCAT integrates k-mer counting and unitig construction, using contextual information to construct globally valid unitigs across datasets. It also improves color encoding by mapping color sets to compact indices, storing differences between consecutive color sets via run-length encoding, both at the individual k-mer level and across unitigs. GGCAT can also incorporate Greedy Matchtigs to further compress unitigs before final storage.

In the following sections, we introduce USTAR-CR (Unitig STitch Advanced constRuction with Colors Reordering), a new greedy heuristic built upon the USTAR2 paradigm, designed specifically for compressing multiple k-mer sets. USTAR-CR extends these principles to colored de Bruijn graphs, enabling fast and memory-efficient construction of compact SPSS representations while also efficiently encoding color information.

2 Method

2.1 Preliminaries

We consider strings composed of characters from the DNA alphabet $\Sigma = \{A, C, G, T\}$. A substring of length k is called a *k-mer*, and we denote its reverse complement by $rc(\cdot)$. Because the originating strand of DNA is often unknown, we treat a

k-mer and its reverse complement as equivalent.

To compress a set of k-mers K, we aim to represent it using a set of longer strings S such that the complete collection of k-mers (and their reverse complements) contained in S exactly recovers K. The *spectrum* of a string set S, denoted $spec_k(S)$, is defined as the set of all k-mers (and their reverse complements) that appear as substrings in any string $s \in S$:

$$spec_k(S) = \{t \in \Sigma^k \mid \exists s \in S \text{ such that } t \text{ or } rc(t) \text{ is a substring of } s\}.$$

The goal is to find a minimal set of strings S such that its spectrum exactly matches that of the original k-mer set K. We formalize this as follows:

Definition 1. *A Spectrum Preserving String Set (SPSS) for a given k-mer set K is a set of strings S, each of length at least k, such that $spec_k(S) = spec_k(K)$.*

A central property of an SPSS is that it compactly encodes the same set of k-mers as the original input, while allowing for flexible string lengths. To assess the efficiency of such a representation, we consider the *cumulative length*:

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

where $|s|$ denotes the length of string s. The following optimization problem thus arises:

Problem 1. *Given a k-mer set K, compute a Spectrum Preserving String Set S that minimizes the cumulative length $CL(S)$.*

This problem can be addressed using graph-based techniques. A k-mer set can be represented as a *de Bruijn graph (dBG)*, where nodes represent k-mers and edges connect overlapping k-mers. It was shown in (Schmidt et al., 2023) that this minimization problem can be solved exactly in polynomial time using an algorithm based on many-to-many minimum-cost path queries combined with minimum-cost perfect matching. The resulting algorithm, *Matchtigs*, guarantees optimal solutions but has a time complexity of $O(n^3m)$, where n is the number of nodes and m the number of arcs. This makes it unsuitable for large-scale datasets.

To overcome these limitations, *USTAR2* (Rossignolo and Comin, 2024b) was introduced as a fast and memory-efficient heuristic. USTAR2 approximates a minimal SPSS by leveraging the connectivity structure of the *compacted de Bruijn graph* produced by BCALM2 (Chikhi et al., 2016), and runs in linear time with respect to the number of nodes/unitigs.

The algorithm proceeds by iteratively selecting seed nodes and extending them into paths as far as possible through unvisited neighboring nodes. This path extension continues until all nodes are included in some path. USTAR2's key innovation lies in its seed and extension heuristics: it selects seed nodes with the highest imbalance (difference between in-degree and out-degree) and prioritizes extending into nodes with fewer connections. This strategy preserves highly connected nodes for future paths, reducing fragmentation and promoting longer paths.

By doing so, USTAR2 significantly reduces cumulative length (CL) by producing fewer, longer strings, thus improving both compression efficiency and downstream query performance.

2.2 USTAR-CR: SPSS with Colors Reordering

USTAR2 compresses individual k-mers set by generating a plain-text representation of the de Bruijn graph. Building on this, USTAR-C extends the method to handle multiple k-mer sets, each linked to a distinct color. USTAR2 begins by constructing the de Bruijn graph for a single k-mer set and compresses the sequences by finding a path cover on the graph. To manage multiple k-mer sets, USTAR-C simply adds a color vector to each node in the graph, which tracks the colors of all k-mers represented by that node. The compression process in USTAR-C follows the same path cover approach as USTAR2, with the added step of merging these color vectors to maintain color information across different sets. An example of the resulting sequences and their color vectors is provided in Table 1.

Sequences	**Color Set Indices (Plain)**	**Color Set Indices (RLE)**
AATAGA	2 2 1 1	2-2 1-2
ACTTCG	4 4 3 3	4-2 3-2
CCAGGC	2 1 1 1	2-1 1-3
CCTCTG	3 3 3 2	3-3 2-1
CTTGAA	3 2 2 2	3-1 2-3

Table 1: Example SPSS with color set indices for $k = 3$. Each sequence is associated with a series of color set indices, which can be compacted using RLE.

Encoding colors for each k-mer in a de Bruijn graph presents two main challenges: (1) efficiently recording all colors associated with each k-mer, and (2) storing this information in a way that minimizes space usage and speeds up processing. To address this, for colored de Bruijn graphs, we incorporate an approach inspired by GGCAT. Instead of storing a separate compressed color bitmap for every k-mer, GGCAT groups colors into color set indices, following a method similar to (Almodaresi et al., 2017). Moreover, each color set is encoded by capturing the differences between consecutive colors and applying Run-Length Encoding (RLE). When writing to disk, the color set indices of consecutive k-mers within each unitig are also run-length encoded. This is highly effective because unitigs tend to be "variation-free," meaning they usually contain only a few distinct color set indices associated with their k-mers.

Consider that USTAR-C has produced a representation of a set of colored k-mers with $k = 3$. Table 1 shows a minimal example of an SPSS representation consisting of five sequences, each of length 6, along with their corresponding color set indices. The colors are grouped into sets identified by an index (see Appendix Table 6). These sequences cannot be further merged or compressed because they do not share overlaps and, equivalently, are not connected within the de Bruijn graph.

However, each color vector can be compressed individually using Run-Length Encoding (RLE), resulting in two runs per sequence and a total of 10 runs, as detailed in the last row of Table 1. The final output of USTAR-C consists of the list of sequences representing the k-mers and a color vector (compressed with RLE) representing the colors associated with all the k-mers.

The k-mers in the previous example cannot be further compressed at the sequence level, as the resulting sequences do not overlap and thus cannot be concatenated. However, some sequences do share the same color patterns, which presents an opportunity to further compress the color information through improved run-length encoding.

Specifically, if two adjacent sequences in the output share the same color at their junction—i.e., the last color of one matches the first color of the next—we can merge their color vectors into a single run, enhancing RLE efficiency. For instance, in Table 1, the sequences CTTGAA and AATAGA have color vectors $(3, 2, 2, 2)$ and $(2, 2, 1, 1)$, respectively. Since the last color of the first vector and the first color of the second are both 2, we can merge them into a single vector: $(3, 2, 2, 2, 2, 2, 1, 1)$, which can be encoded as $(3 - 1, 2 - 5, 1 - 2)$ using RLE. However, this compression is only possible if the sequences are adjacent in the output. In the example shown, this condition isn't met, so the color vectors remain unmerged.

To exploit this compression opportunity, we can reorder the sequences to maximize consecutive matching color sets. Building on this idea, we introduce

USTAR-CR, an extension of USTAR-C that incorporates optimal sequence and colors reordering.

This enhancement introduces the construction of an end-point weight graph (*ewG*) as part of the compression pipeline. In the *ewG*, each sequence is represented as a node with two labeled sides—its first and last color sets. Edges are drawn between node sides that share the same color set, allowing merged color runs for any connected path. This structure enables USTAR-CR to systematically reorder sequences to minimize the number of runs in the final color encoding.

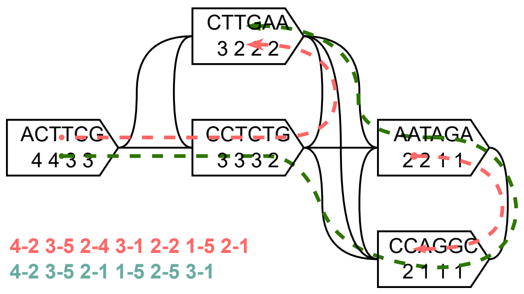


Figure 1: End-point weight graph (ewG) from sequences in Table 1. Red: arbitrary path cover (7 runs). Green: optimized path cover by USTAR-CR (6 runs).

This configuration enables the traversal of paths in the end-point weight graph (*ewG*) to reorder sequences and their corresponding color vectors, thereby allowing adjacent runs to be merged. To maximize such merging while ensuring that each sequence is included exactly once, the goal is to construct long, vertex-disjoint paths. As a result, minimizing the number of color runs translates to solving the problem of finding a minimum vertex-disjoint path cover.

Although an exact solution to a related problem is described in (Pibiri, 2023), it requires additional preprocessing steps that may slow down the compression pipeline. To avoid this, we adopt an efficient greedy approximation strategy, inspired by USTAR (Rossignolo and Comin, 2024a). This method builds a path cover by always selecting the least connected node as the next extension point.

Algorithm 1: USTAR-CR

```
Data: End-point weight graph ($ewG$)
Result: Sequence order $P$
begin
    $P = \emptyset$
    seed-nodes = sort nodes by degree in increasing order
    for $seed \in seed\text{-}nodes$ do
        if seed is not visited then
            visit(seed)
            path = Extend(seed) to the right
            path = Extend(path) to the left
            $P = P \cup \{path\}$
        end
    end
    return P
end
Function Extend(path):
    L = {non-visited neighbors of path head}
    while L not empty do
        $v$ = less connected node in L
        visit($v$)
        path = merge($v$, path)
        L = {non-visited neighbors of $v$}
    end
    return path
```

The pseudo code for USTAR-CR is shown in Algorithm 1. The process starts by sorting nodes in increasing order based on their degree, thereby giving priority to nodes with smaller degree. A non-visited node is then selected as a seed to begin path construction. The path is grown in both directions from this seed by iteratively choosing the least connected, non-visited neighbor as the next node in the path. This approach helps preserve the more connected nodes for later use, reducing the chance of leaving isolated nodes. As nodes are added to the path, they are marked as visited to prevent reuse. Additionally, the orientation (forward or reverse) of each node relative to the path is tracked during construction.

Figure 1 illustrates an example of an end-point weight graph (*ewG*) built from the sequences in Table 1. The red-highlighted path cover represents an arbitrary solution consisting of two paths, resulting in a total of 7 runs. In contrast, USTAR-CR constructs a more efficient path cover (shown in green) by repeatedly extending paths through nodes with the lowest degree. This approach yields a single path that visits all nodes, reducing the total number of runs to 6.

Sequence	**Direction**	**Merged Color Vector (RLE)**
ACTTCG	forward	4-2 3-5 2-2 1-5 2-5 3-1
CCTCTG	forward	
CCAGGC	forward	
AATAGA	reverse	
CTTGAA	reverse	

Table 2: Output of USTAR-CR after sequence reordering. Color vectors are merged and RLE-compressed.

This example demonstrates how leveraging the *ewG* can significantly enhance color compression. Starting from 10 individual runs (as shown in Table 1), the application of an arbitrary path cover reduces this number to 7, while the optimized path cover produced by USTAR-CR brings it further down to 6. The final output, presented in Table 2, reflects this reordering. Because a single path covers all nodes, all color vectors can be concatenated and encoded as one unified run-length-encoded vector, achieving the minimal number of runs.

In the next section, we evaluate USTAR-CR on real sequencing datasets, benchmarking its compression efficiency and runtime performance against state-of-the-art tools.

3 Results

In this section, we compare our proposed methods with the state-of-the-art colored k-mer compression tool, GGCAT. Specifically, we evaluate both GGCAT configured for maximal unitigs and its variant GGCAT GM (which uses greedy matchtigs), alongside our approaches: USTAR-C, which applies plain run-length encoding (RLE) to color vectors, and USTAR-CR, which further improves compression through optimal reordering of color runs.

The benchmarking was performed on a collection of 20 sequencing datasets (detailed in Appendix Table 7), previously used in related studies (Pandey et al., 2018; Rizk et al., 2013; Kokot et al., 2017; Chikhi et al., 2016; Břinda et al., 2021). These datasets span a range of sequencing properties—including paired-end and single-end reads, varying read lengths, and different coverage levels—offering a diverse and representative testing ground for evaluating compression performance under real-world conditions.

To simulate a colored k-mer setting, we merged all datasets and assigned colors to each k-mer based on the input files in which it appears. Appendix Table 8 reports the number of distinct k-mers considered in the evaluation, ranging from 400 million to over 2 billion, depending on the selected k-mer length.

To assess the performance of the evaluated tools, we focused on three primary metrics: Cumulative Length (CL), Sequence Count (SC), and the Number of Runs.

- Cumulative Length (CL): The total length of all compressed k-mer sequences across the datasets, reflecting the effectiveness of sequence-level compression.

- Sequence Count (SC): The number of separate sequences in the output, which serves as an indicator of fragmentation in the compressed representation.

- Number of Color Runs (#runs): The total count of contiguous color segments after encoding, which directly correlates with the efficiency of color compression.

Together, these metrics provide a detailed evaluation of how compact and well-structured the k-mer representations are after compression.

To further compare overall compression performance, we measured the total compressed size, which includes the sequence file compressed using MFCompress (Pinho and Pratas, 2014), the encoded color table, and—for USTAR-C and USTAR-CR—the color vectors compressed with bzip3.

In the following analysis, we present a detailed comparison of all tools based on compression ratio, runtime, and memory consumption.

3.1 Compression of k-mer Sets

In our first experiment, we used the commonly adopted k-mer length of $k = 31$. Table 3 reports the results for all tools prior to compression.

k=31	**GGCAT**	**GGCAT GM**	**USTAR-C**	**USTAR-CR**
CL	6,266,509,634	**3,290,519,704**	3,681,600,490	3,681,600,490
SC	135,191,765	**41,341,022**	43,520,096	43,520,096
#runs	468,952,986	78,730,727	92,771,770	**54,026,538**

Table 3: Comparison of tools before compression using Cumulative Length (CL), Sequence Count (SC), and Number of Runs (#runs). GGCAT GM achieved the lowest CL and SC, while USTAR-CR yielded the fewest runs.

As expected, GGCAT (which generates maximal unitigs) performed the worst across all metrics, yielding the highest cumulative length, sequence count, and number of runs—indicating significant redundancy in both sequences and associated color vectors.

GGCAT GM, which applies a greedy matchtig strategy, significantly improved compression performance—reducing CL by 47%, SC by 69%, and the number of runs by 83%. USTAR-C also delivered substantial improvements, with reductions of 41%, 67%, and 80% in CL, SC, and runs, respectively.

While USTAR-C and USTAR-CR produced identical results for CL and SC, USTAR-CR dramatically reduced the number of runs by 88%, demonstrating the benefit of optimal sequence reordering in compressing color runs.

In summary, GGCAT GM achieved the smallest CL and SC, indicating strong sequence compression, whereas USTAR-CR attained the lowest number of runs, showcasing the effectiveness of RLE-based color compression through path-based reordering.

k=31	**GGCAT**	**GGCAT GM**	**USTAR-C**	**USTAR-CR**
sequences	3,198,171,033	1,028,520,445	915,769,659	908,641,367
colors	–	–	98,742,763	50,959,415
colors table	50,281	50,318	50,281	50,281
total compression	3,198,221,314	1,028,570,763	1,014,562,703	**959,651,063**

Table 4: Tools comparison: considering the sequences file compressed with MF-Compress (*sequences*), the colors file compressed with bzip3 (*colors*), the colors table, and the total compression. All the measures are expressed in bytes. USTAR-CR excelled in all the metrics, followed by USTAR-C and GGCAT GM.

The compression results for each method are presented in Table 4. As anticipated, GGCAT generated the largest FASTA file. GGCAT GM reduced the sequence file size by approximately one-third. In contrast, USTAR-C and USTAR-CR achieved further reductions by offloading color information to a separate file and compressing it independently from the sequences. Thanks to its lower number of runs, USTAR-CR yielded a significantly smaller color file, resulting in the most efficient overall compression.

In total, USTAR-CR reduced the final file size by 70% compared to GGCAT, and by 6.7% compared to GGCAT GM, demonstrating the effectiveness of colors reordering and run-length encoding in optimizing compression.

3.2 Compression: Different k-mer Lengths

In the next section, we examine how varying the k-mer length influences the compression of colored k-mers. Changing the value of k affects both the total number of k-mers and the connectivity between them. In general, smaller k values lead to a denser graph structure due to an increased number of overlaps (refer to Appendix, Table 8).

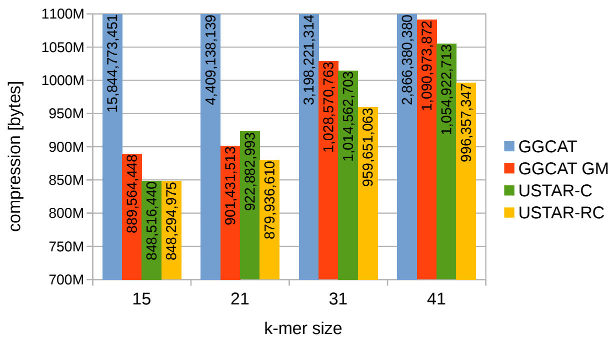


Figure 2: Total compression varying the k-mer size.

The tools were tested using $k \in 15,\ 21,\ 31,\ 41$, and the resulting compression performance is shown in Figure 2. Across all k values, GGCAT consistently delivered the weakest compression. For $k = 15$, USTAR-CR achieved the best compression, followed by USTAR-C and then GGCAT GM. A similar trend is observed for $k = 21$, with USTAR-CR again providing the most compact representation. Notably, the gap between USTAR-CR and USTAR-C widens, emphasizing the importance of run-length encoding (RLE) reordering for optimal compression. At $k = 41$, USTAR-CR once again led in performance, followed by USTAR-C and GGCAT GM. In this case, USTAR-CR delivered the largest improvement over GGCAT GM, reducing the total size by 8.67%.

3.3 Compression: Different Number of Colors

In the previous section, we identified GGCAT GM as the primary competitor to USTAR-CR. We now evaluate how both tools perform as the number of colors —i.e., the number of input files— increases. To do this, we subsampled the dataset with color counts $c \in 10,\ 15,\ 20$ using $k = 31$ (refer to Appendix Table 9). As c increases, the number of k-mers also grows, and the diversity of color subsets rises exponentially, making colored k-mer compression increasingly difficult. The results of this evaluation are presented in Table 5.

k=31	c=10		c=15		c=20	
	GGCAT GM	USTAR-CR	GGCAT GM	USTAR-CR	GGCAT GM	USTAR-CR
# color runs	36,587,632	26,418,625	40,832,766	29,116,036	78,730,727	54,026,538
total compression	433,708,310	**410,570,603**	495,357,773	**464,890,012**	1,028,570,763	**959,651,063**

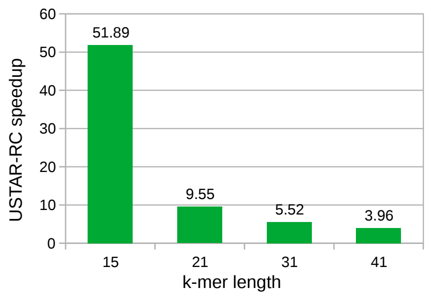
Table 5: Number of runs (#runs) and total compression varying the number of colors (c).

USTAR-CR consistently achieved the lowest number of runs, resulting in the best overall compression across all tested numbers of colors. The greatest difference between GGCAT GM and USTAR-CR occurred at $c = 20$, where USTAR-CR reduced the number of color runs by 31.4% and improved total compression by 6.7%.

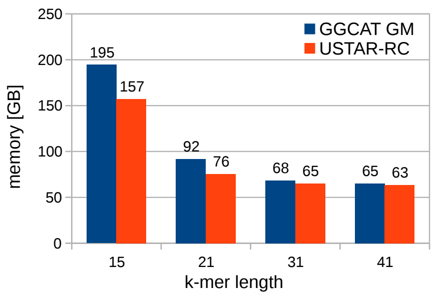
3.4 Time and Memory Usage

In this section, we evaluate and compare the execution time and memory consumption of the top-performing compression tools, GGCAT GM and USTAR-CR.

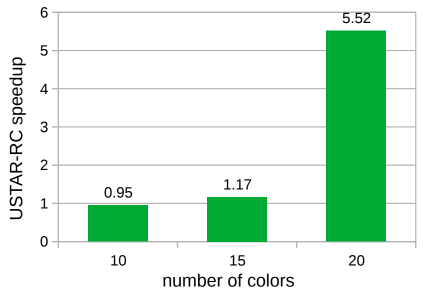
In Figures 3 (a) and (b), we examine execution performance across different k-mer lengths used for compression. The speedup of USTAR-CR compared to GGCAT GM ranges from 3.92× up to 51.89×. The highest speedup occurs at $k = 15$, where USTAR-CR is 51.89× faster than GGCAT GM, while using $157GB$ of memory—about 19.2% less than GGCAT GM. A similar pattern is seen in memory usage, with USTAR-CR consistently requiring fewer resources than GGCAT GM. Figures 3 (c) and (d) show performance comparisons as the number of colors (i.e., k-mer sets) varies. We observe that USTAR-CR's



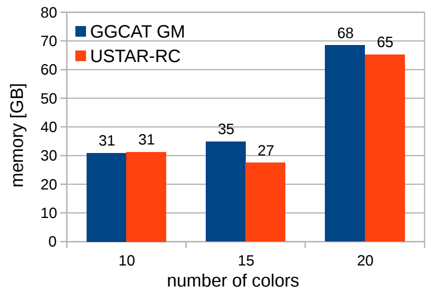
(a) Speedup varying k-mer length.



(b) Memory varying k-mer length.



(c) Speedup varying the number of colors.



(d) Memory varying the number of colors.

Figure 3: Speed up and memory requirement of USTAR-CR with respect to GGCAT GM.

speed advantage over GGCAT GM grows with an increasing number of colors. Memory usage is generally lower for USTAR-CR as well, except when handling 10 colors, where memory requirements are roughly the same.

Overall, USTAR-CR consistently delivers faster execution and lower resource consumption. Notably, USTAR-CR is currently single-threaded, whereas GGCAT GM employs multithreading, suggesting there is room for further optimization of our tool.

3.5 Human reads dataset

In this section, we evaluate the compression quality and performance of GGCAT GM and USTAR-CR on large-scale datasets. For this purpose, we selected the

Human Read dataset from the Genome In A Bottle Consortium (HG004_NA24143_mother)[2] which includes 35 colors and contains over 7.2 billion 31-mers.

Following the approach in Section 3.3, we fixed $k = 31$ and varied the number of colors from 20 to 35. Figure 4 illustrates the speedup and compression ratio of USTAR-CR compared to GGCAT GM. We observe that as the number of colors c increases, USTAR-CR's speedup rises from 9.6× to 64.2×. Meanwhile, the compression ratio gradually decreases from 0.99 to 0.90, indicating a modest increase in file size in exchange for substantially faster processing. Detailed absolute values for all tools can be found in the Appendix, Tables 10 and 11.

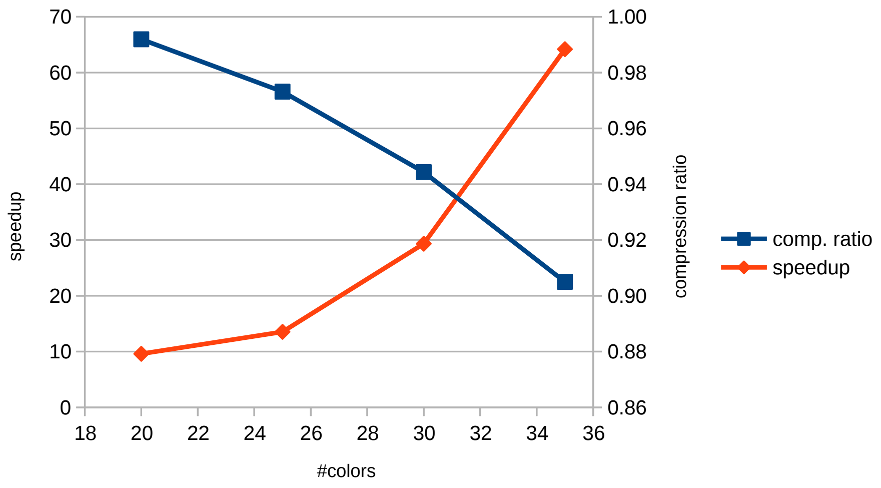


Figure 4: Results related to the human reads dataset. In the Figure, the speedup and the compression ratio change with the number of colors while $k = 31$.

4 Conclusions

In this article, we present USTAR-CR, a novel algorithm designed for efficient compression of multiple k-mer sets. USTAR-CR leverages node connectivity in

[2] https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/AshkenazimTrio/HG004_NA24143_mother/NIST_Illumina_2x250bps/reads/

the colored de Bruijn graph to produce a more compact plain-text representation and employs an optimized encoding scheme for k-mer colors.

Our comparative analysis against GGCAT and GGCAT GM demonstrates that USTAR-CR outperforms these tools in both compression effectiveness and resource usage. For the widely used k-mer length of 31, USTAR-CR achieved the lowest number of runs (#runs) through colors reordering, significantly reducing color redundancy and resulting in compressed files that are 70% smaller than those from GGCAT and 6.7% smaller than GGCAT GM. This advantage holds across different k-mer lengths, with USTAR-CR showing particular strength at lower k values where graph density increases. Furthermore, as the number of colors grows, USTAR-CR consistently delivers the fewest runs and the best compression ratios.

Regarding speed and memory consumption, USTAR-CR proved to be highly efficient, achieving up to a 51.89× speedup over GGCAT GM at $k = 15$, while maintaining significant performance gains across all tested values of k. Its memory footprint is also consistently lower, especially for smaller k values, making it a resource-friendly choice. In summary, USTAR-CR surpasses existing state-of-the-art methods by providing a fast, memory-efficient, and highly compressed solution for representing colored k-mer sets.

Acknowledgments

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

Almodaresi, F., Pandey, P., and Patro, R. Rainbowfish: A Succinct Colored de Bruijn Graph Representation. In Schwartz, R. and Reinert, K., editors, *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*, volume 88 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-050-7.

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

Andreace, F., Lechat, P., Dufresne, Y., and Chikhi, R. Comparing methods for constructing and representing human pangenome graphs. *Genome Biology*, 24, 2023.

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.

Bray, N. L., Pimentel, H., Melsted, P., and Pachter, L. S. Near-optimal probabilistic rna-seq quantification. *Nature Biotechnology*, 34:525–527, 2016.

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

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

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

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

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

Cracco, A. and Tomescu, A. Extremely fast construction and querying of compacted and colored de bruijn graphs with ggcat. *Genome research*, 05 2023.

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

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

Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., and McVean, G. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 44:226–32, 02 2012.

Khan, J., Kokot, M., Deorowicz, S., and Patro, R. Scalable, ultra-fast, and low-memory construction of compacted de bruijn graphs with cuttlefish 2, 12 2021.

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

Luhmann, N., Holley, G., and Achtman, M. Blastfrost: fast querying of 100,000s of bacterial genomes in bifrost graphs. *Genome Biology*, 22, 01 2021.

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

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

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

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

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

Pibiri, G. E. On weighted k-mer dictionaries. *Algorithms for Molecular Biology*, 18(1):3, 2023.

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

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

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

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

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

Rossignolo, E. and Comin, M. Enhanced compression of k-mer sets with counters via de bruijn graphs. *Journal of Computational Biology*, 31(6):524–538, 2024a.

Rossignolo, E. and 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*, pages 368–378. INSTICC, SciTePress, 2024b. ISBN 978-989-758-688-0.

Rossignolo, E. and Comin, M. A linear algorithm for efficient representation of k-mer sets using de bruijn graphs. *Communications in Computer and Information Science*, (24):to appear, 2025a.

Rossignolo, E. and Comin, M. Fast and succinct compression of k-mer sets with plain text representation of colored de bruijn graphs. In *International Conference on Computational Advances in Bio and Medical Sciences*, Singapore, 2025b. Springer Nature Singapore.

Schmidt, S., Khan, S., Alanko, J. N., Pibiri, G. E., and Tomescu, A. I. Matchtigs: Minimum plain text representation of k-mer sets. *Genome Biology (Online)*, 24, 2023.

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

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

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

Zekic, T., Holley, G., and Stoye, J. *Pan-Genome Storage and Analysis Techniques*, pages 29–53. Springer New York, New York, NY, 2018. ISBN 978-1-4939-7463-4.

Appendix

indices	**color sets**
1	1, 2
2	1, 4, 5
3	2, 3
4	3, 4, 5

Table 6: Look-up table for color sets indices. GGCAT, GGCAT GM, USTAR-C, and USTAR-CR associate each k-mer to a color set to save space. Note that not all possible color sets are represented.

dataset	**#15-mers**	**#21-mers**	**#31-mers**	**#41-mers**	**file size [GB]**
SRR001665_1	13,889,837	14,286,068	10,343,472	–	1.04
SRR001665_2	16,371,558	16,895,362	12,058,109	–	1.04
SRR061958_1	225,788,025	388,490,798	404,149,685	392,492,657	0.98
SRR061958_2	265,935,616	482,235,278	495,804,915	475,405,235	0.98
SRR062379_1	109,810,585	152,875,155	160,692,477	160,746,342	5.48
SRR062379_2	108,958,432	151,987,994	159,905,793	158,802,318	5.48
SRR10260779_1	84,250,397	113,667,728	123,624,245	127,090,699	3.94
SRR10260779_2	93,032,179	128,074,943	139,633,894	143,150,103	3.94
SRR11458718_1	89,998,269	126,431,861	137,995,280	143,397,012	6.96
SRR11458718_2	94,018,791	134,997,414	150,549,990	159,144,668	6.96
SRR13605073_1	43,488,336	54,085,000	55,764,573	54,682,553	0.97
SRR14005143_1	11,307,338	13,223,059	15,005,192	16,272,583	0.24
SRR14005143_2	23,691,810	28,456,533	31,850,681	33,872,511	0.24
SRR332538_1	10,624,064	11,404,027	11,382,816	10,666,430	1.22
SRR332538_2	18,741,106	25,674,930	28,880,136	27,477,871	1.22
SRR341725_1	132,442,790	188,913,254	185,618,107	176,391,089	1.93
SRR341725_2	136,484,353	196,035,961	192,133,588	181,970,438	1.93
SRR5853087_1	159,744,051	316,438,109	382,773,071	399,026,650	7.79
SRR957915_1	126,236,121	208,110,514	239,200,400	250,988,377	4.35
SRR957915_2	188,867,779	335,926,750	364,597,018	361,352,380	4.35

Table 7: List of datasets used in the experiments, taken from previous studies on k-mers compression (Pandey et al., 2018; Rizk et al., 2013; Kokot et al., 2017; Chikhi et al., 2016; Břinda et al., 2021). Each dataset is represented by a color from 0 to 19. There are 2^{20} possible color subsets. Note that SRR001665_1 and SRR001665_2 do not have any 41-mers.

datasets	**#k-mers**
k=15	453,004,928
k=21	1,985,116,586
k=31	2,210,756,684
k=41	2,245,074,159

Table 8: The number of distinct k-mers extracted from 20 reads files (shown in Table 7) for different k-mer length.

datasets	**#31-mers**	**#color subsets**
c=10	920,328,720	1,024
c=15	1,047,860,309	32,768
c=20	2,210,756,684	1,048,576

Table 9: Number of 31-mers and color subsets as the number of colors (c) varies. As c grows, the number of k-mers increases while the number of color subsets rises exponentially, making the datasets difficult to compress.

compression	**#colors**	**GGCAT**	**GGCAT GM**	**USTAR-CR**	**ratio**
k=31	35	14,936,284,972	4,432,771,740	4,898,121,318	0.90
	30	8,865,599,517	3,013,844,429	3,191,468,842	0.94
	25	5,216,850,645	2,024,431,962	2,080,235,709	0.97
	20	4,137,736,511	1,565,672,696	1,578,400,980	0.99

Table 10: Human reads dataset compression ratio.

CPU-time	**#colors**	**GGCAT**	**GGCAT GM**	**USTAR-CR**	**speedup**
k=31	35	28,397	4,302,924	67,034	64.19
	30	15,769	1,059,323	36,101	29.34
	25	7,241	211,815	15,650	13.53
	20	5,334	128,924	13,426	9.60

Table 11: Human reads dataset speedup.