Journal Title Here, 2022, pp. 1–12

doi: DOI HERE
Advance Access Publication Date: Day Month Year
Paper

PAPER

Tinted de Bruijn Graphs for efficient read extraction from sequencing datasets

Léa Vandamme,[1,*] Bastien Cazaux[1] and Antoine Limasset[1,*]

[1]UMR9189 CRIStAL, Univ Lille, CNRS, Centrale, F-59000, Lille, France
[*]Corresponding author. lea.vandamme@univ-lille.fr
FOR PUBLISHER ONLY Received on Date Month Year; revised on Date Month Year; accepted on Date Month Year

Abstract

The study of biological sequences often relies on using reference genomes, yet achieving accurate assemblies remains challenging. Consequently, de novo analysis directly from raw reads, without pre-processing, is frequently more practical. We identify a very commonly shared need across various applications: identifying reads containing a specific kmer in a dataset. This kmer-to-reads association would be pivotal in multiple contexts, including genotyping, bacterial strain resolution, profiling, data compression, error correction or assembly. While this challenge appears similar to the extensively researched colored de Bruijn graph problem, resolving it at the read level would be prohibitively resource-intensive in practical applications. In this work, we demonstrate its tractable resolution by leveraging certain assumptions for sequencing dataset indexing. To tackle this challenge, we introduce the Tinted de Bruijn Graph concept, an altered version of the colored de Bruijn graph where each read within a sequencing dataset represents a unique source. We developed K2R, a highly scalable index that implement such search efficiently within this framework. K2R's performance, in terms of index size, memory footprint, throughput, and construction time, is benchmarked against leading methods, including hashing techniques (e.g., Short Read Connector) and full-text indexing (e.g., Spumoni and Movi), across various datasets. K2R consistently outperforms contemporary solutions in most metrics and is the only tool capable of scaling to larger datasets. To prove K2R scalability we indexed two human datasets of the T2T consortium: the 126X coverage ONT dataset was indexed in 18hours using 19GB of RAM for a final index of 9.5GB and the 56X coverage HiFi dataset was constructed in 90 minutes using 5Gb of RAM for a final index of 207Mb. The K2R index, developed in C++, is open source and available on Github `github.com/LeaVandamme/K2R`.

Key words: Indexing, Sequencing datasets, Compression

Introduction

The pursuit of direct and unbiased access to complete nucleic sequences stands at the forefront of sequence analysis fields. This quest, although significantly advanced over the last two decades, still presents substantial challenges in obtaining reference sequences. A primary obstacle lies in the complexity of assembly processes, both theoretically [28] and practically [30]. These processes are often hindered by the intrinsic nature of the data and may not directly address key biological questions, such as quantification or dataset comparison. Consequently, the *de novo* analysis of raw datasets holds paramount importance.

Since the inception of the seed and extend paradigm [4], numerous applications have predominantly relied on the identification of fixed-length word matches, commonly referred to as *kmers*, for detecting sequence similarities. An efficient way to represent a kmer set from sequencing datasets is through the use of a de Bruijn graph [10], which retains, to a certain extent, the original ordering of kmers. Kmer sets have found widespread application in various domains, including quantification [39], dataset comparison [34], phylogeny [23], assembly [6, 5], compression [8], correction [20], and alignment [22, 12]. The advent of short-read sequencing technologies has particularly emphasized the extensive utility of the de Bruijn graph as a convenient and effective structure. As a result, Kmer indexes have become well-researched structures, boasting highly efficient implementations either based on hashing [25, 32] or full-text [2, 36] methods.

However, a notable limitation of such indexes is their loss of significant information, specifically the colocalization of kmers within common DNA fragments. In the context of short reads, this information loss was not deemed critical, as the kmer size (k) was generally comparable to the size of the reads. In contrast, for long reads, this loss becomes more consequential as it results in the omission of extremely valuable long-distance information [13]. A potential solution involves the ability to identify the specific reads in which a given kmer appears. Full-text indexes [18], which can locate occurrences of patterns within their indexed datasets, offer a natural solution. Recent advancements in BWT-based full-text indexes have led to the development of two novel structures that

© The Author 2022. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

enhance the previously dominant FM index: the r-index [7], a full-text index occupying space proportional to $\mathcal{O}(r)$, where r represents the number of BWT runs for the input text of size n. This index introduces a suffix array sampling strategy requiring only $2r$ space, significantly reducing the locate time from $\Omega(n/r)$ per occurrence (as seen in traditional FM-indexes) to $\mathcal{O}(\log(n/r))$. More recently, Movi [40], based on the move index [29], achieves both $\mathcal{O}(r)$ space efficiency and $\mathcal{O}(1)$-time queries. Nonetheless, these structures face challenges in compressing the high redundancy of reads, which often contain noise that disrupts redundancy and introduces irrelevant novel sequences to index. Our work aims to evaluate the efficiency of such indexes in relation to dataset properties, such as genome size, coverage, and error rates.

This problem bears a strong resemblance to the extensively studied colored de Bruijn graph issue [14]. A colored de Bruijn graph, constructed from a collection of documents, associates each kmer with the list of documents containing it. One could conceptualize each read as an individual document to rely on a colored de Bruijn graph implementation. However, this approach becomes prohibitively expensive given the vast number of reads anticipated from gigabase level genomes sequencing, easily reaching tens of millions when indexing ten thousands dataset is already very expensive [27, 24]. Despite this, we emphasize that assumptions about the kmer distribution in colored de Bruijn graphs are often limited. In contrast, we aim to demonstrate the problem tractability of indexing a sequencing dataset by leverage specific assumptions.

In this work, we outline these properties and propose straightforward techniques to incorporate them. For practical purposes, we introduce a scalable and rapid tool named K2R (Kmer to Reads), which leverages sequencing datasets properties, and perform an extensive benchmark against state-of-the-art structures capable of executing similar tasks. The K2R index, developed in C++, is open source and available on Github `github.com/LeaVandamme/K2R`

Methods

Notation and problem

In the following, all the strings will be defined in the DNA's alphabet (an alphabet of size 4), but all the results could be generalized to a finite alphabet. A kmer of a string r, is a substring of r of length k. For a positive integer k, the *k-spectrum* of r is the set of all the kmers of r. For an integer $m \leq k$ and a specific total order of the set of the m-mers, the *m-minimizer* (or just minimizer [35]) of a kmer x, denoted by $m(x)$ is the smallest string of the *k-spectrum* of x according to a given order.

To define the set of similar reads $R' \subseteq R$ (where R is the initial set of reads) to a sequence s, we choose to use the shared kmers number as a filter mimicking [37]. A read r will then be considered similar to s if the number of shared kmers with s is greater than a threshold t, i.e. $|k\text{-spectrum}(r) \cap k\text{-spectrum}(s)| \geq t$.

Tinted de Bruijn graph

To support multiple queries on the same set of reads, we need to store for each kmer x, the set of reads where it appears, i.e. $\{r \in R : x \in k\text{-spectrum}(r)\}$. As a de Bruijn Graph can be viewed through a set of kmers (by storing arcs, or by testing 4 extensions (alphabet size) to access to possible successive kmers), these two notions are commonly conflated.

Given a document collection $\mathcal{F}$ (sequencing datasets or assembled genomes), which is a cover for the sets of all reads R ($\cup_{F \in \mathcal{F}} F = R$), a *colored de Bruijn graph* is defined as a de Bruijn graph where each kmer is associated to the documents that contain it, i.e. a mapping from $\cup_{r \in R} k\text{-spectrum}(r)$ to $\mathcal{P}(\mathcal{F})$. $\mathcal{F}$ is the set of the fasta files, and each kmer is associated with the list of fasta files where it appears.

Similar to the colored de Bruijn Graph, we introduce the *Tinted de Bruijn graph* which is a de Bruijn graph where each kmer is associated to the reads that contain it, i.e. a mapping from $\cup_{r \in R} k\text{-spectrum}(r)$ to $\mathcal{P}(R)$. The Tinted de Bruijn graph can be seen as a special case of colored de Bruijn graph where each read from a sequencing dataset is seen as a distinct source.

We first remark that such structure could, to some extend, recover its original dataset by navigating the de Bruijn graph and reconstructing the reads. However in the case of duplicated region within a given read we can obtain several possible distinct "assemblies" of such reads (multiple Eulerian paths in the induced subgraph of the de Bruijn graph by these reads). This mean that the Tinted de Bruijn graph is not reversible (impossibility to turn up the initial reads and thus some information is lost). Since the presense of repeated kmers corresponds to these non-unique assemblies, by storing, in extra space, the different positions in each read of these kmers, we obtain a reversible version of the *Tinted de Bruijn graph*, which we call the *Reversible Tinted de Bruijn graph.* This reversibility property makes this Reversible Tinted de Bruijn Graph an interesting new element of compressed full-text data structure family, as BWT and FM-index.

Instead of storing the Tinted de Bruijn graph which can be costly, we focus in this work on a lossy version of the Tinted de Bruijn graph, called *Minimizer Tinted de Bruijn graph*, where minimizers are used to represent the kmers as each miminimizer m is associated with the reads that contains a kmer that has m as minimizer, i.e. $\{r \in R : m \in \{m(y) : y \in k\text{-spectrum}(r)\}\}$.

As for all kmer x, $\{r \in R : x \in k\text{-spectrum}(r)\} \subseteq \{r \in R : m(x) \in \{m(y) : y \in k\text{-spectrum}(r)\}\}$, the Minimizer Tinted de Bruijn graph can have false positives but no false negatives. In addition, the scattered positions of the minimizers and the minimizers shared between different kmers mean that this structure is not reversible. We also pinpoint that the Minimizer Tinted de Bruijn graph is quite inefficient to navigate contrary to the regular Tinted de Bruijn graph.

In the following, we present our efficient implementation the Minimizer Tinted de Bruijn graph dubbed K2R (see Fig. 1).

Overview

Given the conceptual similarity between the Tinted de Bruijn graph and the Colored de Bruijn graph, a naive implementation of a Tinted de Bruijn graph would involve an associative structure, such as a hashmap, linking kmers to the list of read identifiers that contain them. Interestingly, an initial tool, SRC [26], developed for short reads, actually implements this strategy using an efficient Minimal Perfect Hash Function (MPHF) [21] as the associative structure. However, we argue that the cost of such a structure would be prohibitively high for large datasets. For example, a human long-reads dataset of 10 million 10kb reads (approximately 33X coverage) would represent more than 3 billion kmers requiring 24GB of storage (for $k = 31$), and each kmer would be associated with dozens of read IDs, necessitating hundred bytes per association. This would mean several hundreds gigabytes of storage without any overhead and without accounting for sequencing errors. In

this section, we describe the assumptions we can make when indexing a sequencing dataset and how we can exploit such properties to optimize the naive solution.

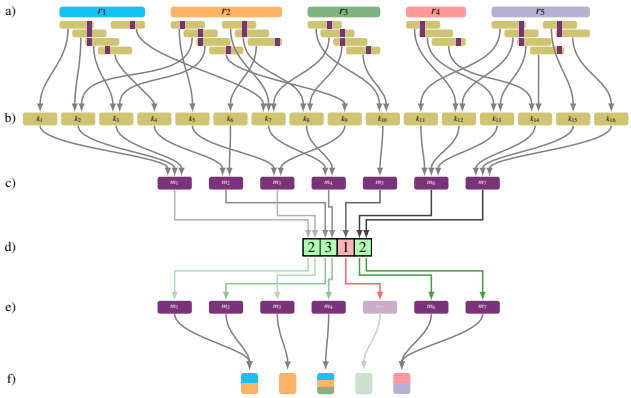

Fig. 1. Example of our lossy version of the Tinted de Bruijn Graph. For a set of reads (a), we can extract the set of kmers (b). Each kmer (b) is linked to a minimizer (c) with possible collisions. By using counting filter (d), we can remove weak minimizer (e). Each remaining minimzer (e) is linked to a color (f).

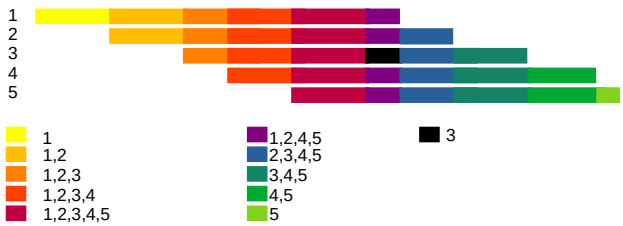

Fig. 2. Simple example of how overlaping reads create subsequences whose kmers are the same colors. We also display the impact of a sequencing errors in the black region that create read specific sequences.

Sparse signal

The primary challenge in scaling colored de Bruijn graphs arises from the inherent difficulty of making a priori assumptions about the kmer distribution within a dataset collection [3]. In contrast, sequencing datasets predominantly target sequences significantly larger than a kilobase, with relatively low coverage that rarely exceeds a hundredfold. This coverage expectation implies that most kmers will be present in a number of reads closely matching the coverage level, marking their presence in an exceptionally small fraction (by several orders of magnitude) of the total reads, as even bacterial genome sequencing can yield millions of reads. Consequently, unlike with colored de Bruijn graphs, it becomes viable to employ sparse encoding rather than bitvector representations to efficiently denote the presence of kmers in reads.

Employing a 32-bit identifier list to represent kmers turns out to be prohibitively expensive in terms of memory usage, demanding hundreds of bytes per kmer for a standard whole-genome sequencing dataset. A more efficient alternative involves using a sorted list with delta encoding, which significantly reduces the memory footprint. Initially, one might assume that applying compression techniques at such a scale could detrimentally affect construction or query times. However, advancements in compression technology have led to the development of highly optimized algorithms [38]. These algorithms leverage SIMD (Single Instruction, Multiple Data) instructions, enabling near-state-of-the-art compression efficiency with minimal impact on overall throughput. Hereafter, we will refer to a (compressed) list of read identifiers associated with a kmer as a color. As example for a HiFi *E. coli* sequencing with 200X coverage the integers list goes from 1,083MB to 133MB a 8-fold improvments without wall-clock drawback.

Redundant colors

An important observation in sequencing datasets is the considerable proportion of overlap among reads, which is directly related to the level of coverage. For instance, with 10X coverage, one can anticipate that reads will overlap by approximately 90% of their length, while at 100X coverage, the overlap can increase to 99%, and so forth. In this context, colors in our dataset predominantly represent combinations of reads that overlap, covering specific genomic regions, as illustrated in Figure 2. Consequently, the diversity of distinct colors observed is relatively limited, since each new read starting position (and consequently, each end position) generates a novel combination. Assuming an ideal scenario devoid of sequencing errors, the number of distinct colors would be expected to increase linearly with the number of reads, necessitating approximately $\mathcal{O}(\log(N))$ bits to encode colors for N reads. However, the scenario becomes more complex with the presence of kmers repeated across the genome or kmers that contain sequencing errors. Sequencing errors introduce a significant number of additional colors in two ways: firstly, by creating singleton colors unique to individual reads, and secondly, by generating 'incomplete' colors that resemble existing colors but with 'gaps' due to the absence of one or more reads, attributable to missing kmers caused by sequencing errors. This leads to the observation that a substantial number of kmers actually share the same color, a property determined by the coverage, read length, and sequencing error rate.

To leverage this characteristic efficiently, we have developed a strategy that indexes only distinct colors by associating them with unique color identifiers, thereby avoiding the redundancy of storing numerous identical colors. This approach entails maintaining a table that links each kmer to its corresponding color identifier, and another table that maps each identifier to the actual list of read identifiers. This method enables the storage of each distinct list precisely once, albeit at the cost of a minor overhead due to the inclusion of identifier integers.

Succesive kmers

Upon closer inspection, we can observe more specifically that overlapping kmers from sequence reads are frequently assigned the same color, primarily because they originate from identical genomic regions and are thus present in the same reads (see Figure 2). Theoretically, one could attempt to reconstruct such regions and map kmers to their respective regions, which would reflect their shared colors; however, this process is anticipated to be nearly as resource-intensive as genome assembly.

An alternative, more dynamic approach to leverage this characteristic involves the utilization of minimizers. Minimizers, which are the minimal m-mers within kmers, are often shared among overlapping kmers [33], leading to a

significantly reduced number of unique minimizers compared to the total number of kmers in a dataset. Ideally, a scheme would select a unique minimizer for every group of $k - m$ kmers. Through the application of "random minimizers", which employ a hashing function, it is estimated that the number of selected minimizers required is twice the minimal theoretical number. Nevertheless, by adopting advanced minimizer selection algorithms [41, 31], onecan surpass these expectations and further reduce the number of selected minimizers in practice.

However, a notable limitation of this technique is the inherent association of all kmers sharing a minimizer with the same color, potentially leading to false positives. To avoid the incidence of false negatives, each minimizer is associated with every read in which it appears. This methodology may result in distinct kmers, which occur in separate reads without shared kmers, logically possessing distinct colors, being assigned the same minimizer and consequently, the same color. Such scenarios, albeit infrequent, could inadvertently introduce false positives, adversely affecting downstream analyses. Conversely, this propensity for false positives may inadvertently counteract the generation of "subcolors" due to sequencing errors, as previously discussed, by incorrectly assigning some erroneous kmers to the correct color, where the genuine genomic kmer would be assigned. We posit that these false positives can be efficiently managed by calculating and reporting the actual number of shared kmers when the selected reads are analyzed, thus providing a balanced approach to minimizing data misinterpretation while maximizing scalability.

Handling sequencing errors

A fundamental obstacle in addressing our problem is the occurrence of sequencing errors, which introduce novel sequences (and kmers). These are predominantly unique, rendering them incompressible and biologically irrelevant. Moreover, these errors dilute the overall redundancy by transforming abundant sequences into rare novel variants. Traditional full-text indexing approaches often struggle in these scenarios, as they are designed to index entire texts indiscriminately and are typically employed on assembled sequences "cleaned" of sequencing errors for this reason.

In contrast, the success of the de Bruijn graph approach can be partly attributed to its ability to manage sequencing errors effectively by leveraging the concept of kmers abundance. In practice, kmers containing sequencing errors are expected to occur infrequently, whereas the abundance of genomic kmers is closely aligned with sequencing coverage. Thus, implementing a simple abundance filter that retains "solid" kmers appearing more than T times in the dataset and excluding "weak" kmers that appear fewer than T times can eliminate nearly all erroneous kmers with minimal loss of relevant kmers, provided the threshold T is optimally set according to the observed abundance [11].

Since we focus on indexing minimizers, we introduce a novel variation of this filtering technique applied at the minimizer level. It follows logically that if a kmer is solid (seen more than T times), then its minimizer is also considered solid. Consequently, excluding weak minimizers does not result in the exclusion of solid kmers. However, minimizer-level filtering may not effectively eliminate erroneous kmers that share a minimizer with solid kmers. This leads us to anticipate that minimizer filtering is less precise than kmer filtering, with the benefit of significantly reducing the inherent counting cost, as it involves quantifying an order of magnitude fewer keys.

Implementation Details

Index Structure

In our index structure, we employ a dual mapping system consisting of a [minimizer:(color)identifier] map and an [identifier:color] map, where minimizers and color identifiers are integers, and colors are compressed lists of integers. We opted to use the unordered_dense hash map, available at `github.com/martinus/unordered_dense`, due to its exceptional performance characteristics, which have been leveraged in several bioinformatics tools, including kallisto [9] and mashmap [15].

Upon inserting a read into the index, its minimizers are computed and inserted into the minimizer map. Each corresponding identifier is then updated to incorporate the new read identifier into its color list. If the associated color does not preexist within the map, a new identifier is generated and added to the identifier map, along with its newly associated color. Colors are compressed using the TurboPFor library `github.com/powturbo/TurboPFor-Integer-Compression`, which displays almost state-of-the-art compression levels at a very low computational cost.

To keep the index lean and efficient, the identifier map also tracks the number of minimizers referencing each identifier. When a minimizer's association shifts to a different identifier, the count of the previous identifier is decremented. If this count reaches zero, indicating no remaining minimizers point to it, the identifier and its associated colors are purged from the map. This mechanism prevents the accumulation of orphaned colors, thus optimizing the memory footprint of the index.

Our choice of index structure imbues it with dynamic capabilities, allowing for real-time querying at any point during the read insertion process. While static structures may offer advantages in memory efficiency, they lack flexibility and require a more cumbersome construction phase.

Minimizer Abundance Filtering

To enhance construction efficiency and minimize the index size, we implement an optional filtering mechanism for weak minimizers using a one-hash Counting Bloom filter [16] to approximate minimizer abundance within the dataset. It's important to note that a Counting Bloom filter can only overestimate counts, leading to false positives by misclassifying some weak minimizers as solid due to hash collisions with genuinely solid minimizers.

After approximate minimizer counting, we convert the Counting Bloom filter into a Regular Bloom filter by transforming each cell, originally representing counts with integers, into a single bit. This bit indicates the solidity of the associated minimizer, significantly reducing memory consumption during the actual index construction. Using this strategy, we limit the peak memory usage as the hash tables should become the memory bottleneck using a well sized Bloom filter as shown in Figure S1 of the Appendix.

We also observe the opposite pattern of overabundant minimizers linked to highly repeated regions in a genome, either due to biological mechanisms or sequencing artifacts. Since such minimizers can be associated with an incredibly high number of reads, they can seriously hinder index performance and "pollute" the output with many irrelevant matches. To

counteract this, we implement an optional maximum minimizer abundance filtering, as commonly performed by other tools [19].

When employing minimizers in place of kmers, alongside Counting Bloom filters to sift through them, a natural concern arises regarding the potential influence of false positives introduced by this method on the final output. To address this, we conduct an evaluation illustrated in Figure S2 of the Appendix, where we calculate the proportion of reads selected based on their minimizer content relative to the total number of reads achieving the requisite kmer similarity threshold. Remarkably, a substantial majority of queries exhibit a ratio exceeding 95%, indicating that nearly all the selected reads are, in fact, akin to the query. This high ratio underscores the efficiency of our approach in maintaining a strong correlation between selected reads and their relevance to the query.

Parallelization Overview

The filtering phase is parallelized with a mutex array to secure sections of the Counting Bloom Filters, enabling efficient concurrent operations. Parallelizing the index construction phase is more complex due to dependencies on synchronized maps. We mitigate this by implementing parallelization at the read level, where minimizers are computed across sequence substrings concurrently. Tasks related to color management, including decompression, updates, and recompression, are executed in parallel, protected by mutex arrays to ensure data integrity. This setup allows concurrent modifications to the map array without data corruption. Query operations are fully parallelizable, unlike the construction phase, enabling high-throughput processing.

Minimizer Scheme

Minimizers reduce complexity by representing a set of overlapping kmers with a single m-mer. The ideal minimizer selection scheme would identify one m-mer for every $k - m + 1$ sequence of overlapping kmers. Practical implementations, such as the random minimizer strategy, tend to select approximately twice as many minimizers as the ideal case.

Recent advancements in minimizer selection techniques aim to closely approach this theoretical lower bound, thus reducing the quantity of necessary minimizers. We employ decycling set minimizers [31], which minimize the count of selected minimizers, albeit at the cost of increased computational overhead.

Homocompression

To address homopolymer run length errors prevalent in HiFi and PacBio sequencing reads, we propose an optional 'homocompression' feature. This technique lossily represents sequences within reads by compressing consecutive occurrences of a nucleotide X. . . X into a single instance of X. As highlighted in the La Jolla Assembler paper [5], applying homocompression to HiFi reads can result in a threefold reduction in total error count, making a substantial proportion of reads error-free. This approach not only improves sequencing data accuracy but also enhances the overall reliability of genomic analysis by significantly reducing the impact of homopolymer-associated errors.

Results

All experiments were conducted on a single cluster node equipped with an Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz, 128GB of RAM, and running Ubuntu 22.04. To validate our results on real data, we chose to perform the same analysis on closely related *E. coli* sequencings. We selected a recent ONT sequencing with a low error rate, approximately 3%, (Accession SRR26899125), a 20kb HiFi sequencing (Accession SRR11434954), and a HiSeq X Ten paired-end sequencing (Accession DRR395239). As a medium-sized genome, we also selected two *C. elegans* datasets, one ONT sequencing (Accession SRR24201716) and another HiSeq X Ten paired-end sequencing (Accession ERR10914908).

Finally, to assess K2R scalability, we selected two human datasets from the T2T project. The HiFi datasets (SRX7897685, SRX7897686, SRX7897687, SRX7897688, and SRX5633451) together amount to 56.8X coverage, comprising both 20kb and 10kb libraries. The ONT sequencings, detailed at `github.com/marbl/CHM13/blob/master/Sequencing_data.md`, total 126X coverage with an approximate error rate of 6%.

State of the art

We categorize tools capable of processing Tinted de Bruijn graph queries into two main types. The first type includes full-text indexes, which can locate kmer occurrences within an indexed read file. Utilizing a starting position array enables the determination of specific reads in which a queried kmer is present. The second approach involves kmer indexes, which are based on associative structures like hash functions. Despite significant advancements in reducing the memory footprint of kmer indexing to below 10 bits per kmer [32], managing the read identifier lists remains a primary challenge, as these lists are expected to be larger by several orders of magnitude. To our knowledge, the only specialized tool implementing this strategy is SRC [26], employing an efficient Minimal Perfect Hash Function [21].

Recent developments in BWT-based full-text indexes have introduced two innovative structures enhancing the capabilities of the FM index, which dominated the field for over a decade. The latest tool utilizing the r-index for large-scale queries is SPUmoni [1], while Movi [40] is the sole tool based on the move index, performing comparably to SPUmoni.

As outlined in the introduction, a fundamental limitation of both full-text and kmer-based indexes is their difficulty in managing sequencing errors, leading to the indexing of irrelevant sequences. Conversely, kmer abundance filtering allows kmer indexing methods to efficiently exclude non-essential kmers.

In our benchmarking, both SRC and K2R indexed 31mers and excluded those appearing only once in the dataset, representing the most basic level of filtering. This creates a theoretical disparity in comparisons, as full-text indexes inherently index the entire text. However, in practical terms, users of these techniques have no other options but to employ them as they currently exist, and most analyses actually filter out at least unique kmers as they are considered unreliable.

Index Construction

We first evaluate the cost of constructing an index from whole genome datasets with the different tools. In Figure 3, we assess the time and memory required according to the input coverage on three *E. coli* datasets. We observe that for all tools, both time and memory requirements grow linearly with the input size. However, their respective performance varies greatly. A first observation is that both SRC and K2R are

equally fast, while Movi and Spumoni are significantly slower. This effect is quite small for HiFi data but is amplified for ONT data and even more so for Illumina datasets. Memory-wise, K2R is the most efficient, followed by SRC, then Movi, and finally Spumoni. We performed the same experiment on the 20 times larger *C. elegans* genome in Figure 4 and obtained similar results, highlighting K2R's scalability in both time and memory. As a check we also performed such benchmark on simulated reads from the reference genomes and obtain very similar results displayed in the Figure S3 and S4 of the Appendix. We also performed this analysis on the two human datasets, using exclusively K2R because of scalability issue, that are displayed in the Figure S5 of the Appendix. We highlight that the 126X ONT dataset was indexed in less than 2 days using only 19GB of RAM. More impressively, the 56X HiFi dataset was indexed in less than two hours using 5GB of RAM. These results prove that such indexes are quite tractable to construct on various datasets, even on gigabase-scale genomes, on a modest workstation.

To further analyze the influence of noise, we evaluated in Figure 5 the time and memory requirements according to the input error rate, with coverage fixed at 50X. Since testing such parameters on real data poses challenges, this benchmark was conducted exclusively on simulated reads from the *E. coli* and *C. elegans* reference genomes. This analysis revealed two distinct patterns. The error rate has almost no influence on the construction time for K2R and SRC, while it significantly impacts both Spumoni and Movi. Following the same trend, the error rate has a minor influence on the memory usage of SRC and K2R; it can even decrease with a higher error rate as fewer k-mers are indexed due to abundance filtering. We observe that full-text indexes are quite hindered by the error rate, performing well with a low error rate and becoming costly with a high error rate. If SRC outperforms both BWT-based methods in terms of time and memory, regardless of the error rate, K2R itself is superior in terms of time and memory efficiency.

Index Size

While the construction cost is a key factor in tractability, the actual final index size is more important since the construction needs to be performed only once but the index can be used multiple times. As before, we first assess the impact of coverage in Figure 6 for *E. coli* and in Figure 7 for *C. elegans* datasets.

We observe very different patterns between the different data types. For HiFi data, all tools except SRC perform quite well, using less than 1GB for 200X coverage. However, with ONT data, only K2R manages to do so, and Movi's indexes even become larger than those of SRC. Illumina data follow the same trend but Movi indexes grow very similarly to SRC. The differences between SRC and K2R are even more pronounced with the *C. elegans* data. As a check we also benchmarked simulated reads from *C. elegans* genome and obtain very similar results displayed in Figure S6 of the Appendix. Finally, as a scalability experiment, we report the index size from human datasets in Figure 8. While indexing high-coverage human datasets is relatively inexpensive in practice (requiring less than 10GB, which is more than an order of magnitude smaller than the actual dataset), we find that the HiFi data index is even an order of magnitude smaller, indicating that sequencing errors can significantly impact the size of such indexes.

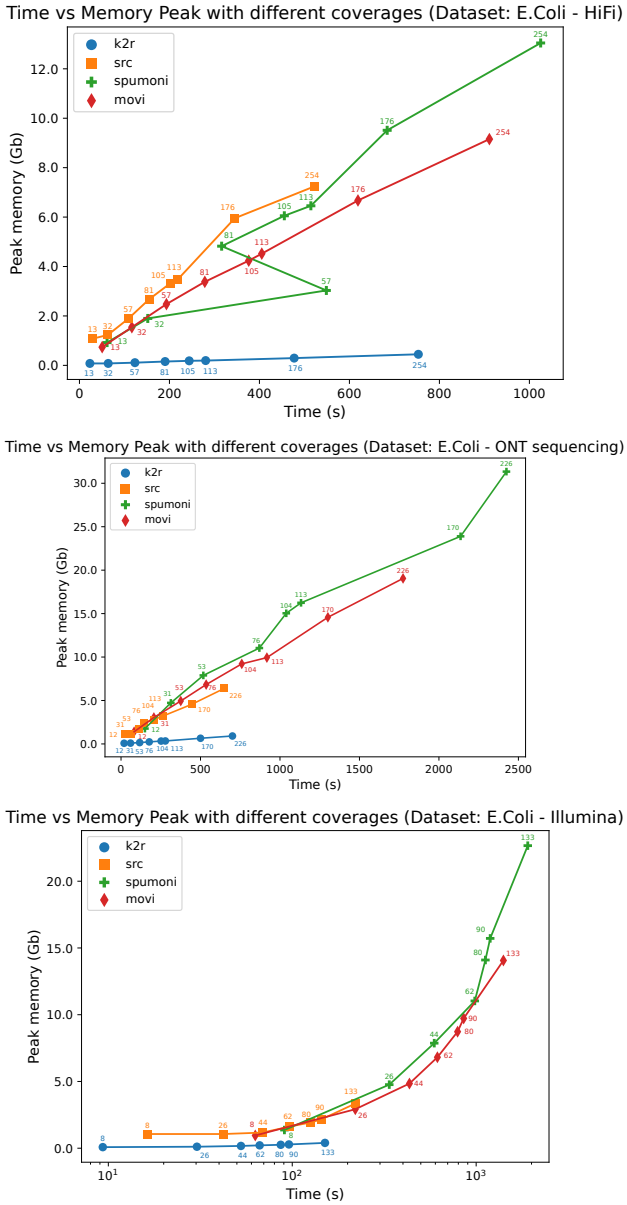

Fig. 3. Memory peak and CPU time used during the index construction with three different kind of data (HIFI (top), ONT (center) and Illumina (bottom)) from *E. coli* genome with varying coverage (specified on the labels). The last graph is in logarithmic scale for readability purposes.

While construction costs are a significant consideration, the final size of the index is more crucial. This is because construction occurs only once, as the index can be utilized multiple times. First, we evaluate the effect of coverage in Figure 6 for *E. coli* and in Figure 7 for *C. elegans* datasets. As expected, the index sizes increase almost linearly with the input size. However, we note distinct behaviors across different data types. For HiFi data, all tools except SRC are efficient, requiring less than 1GB for 200X coverage. In contrast, for ONT data, K2R is the only tool that maintains this efficiency, while Movi's index sizes exceed those of SRC. Illumina data exhibit a similar pattern, with Movi's indexes growing comparably to SRC's. The disparity between SRC and K2R is even more stark with *C. elegans* data. We

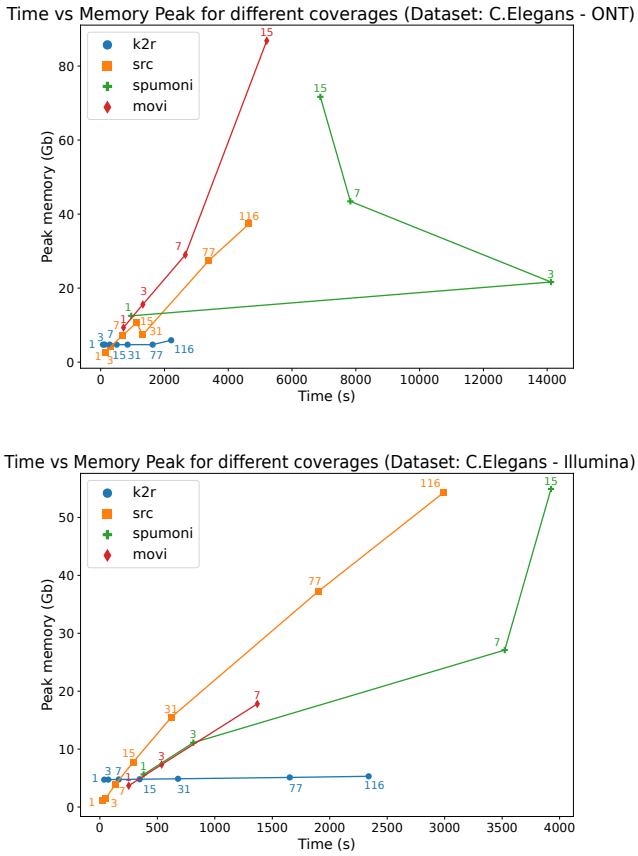

Fig. 4. Memory peak and CPU time used during the index construction with two different kind of data (ONT (top) and Illumina (bottom)) from *C. elegans* genome with varying coverages (specified on the labels).

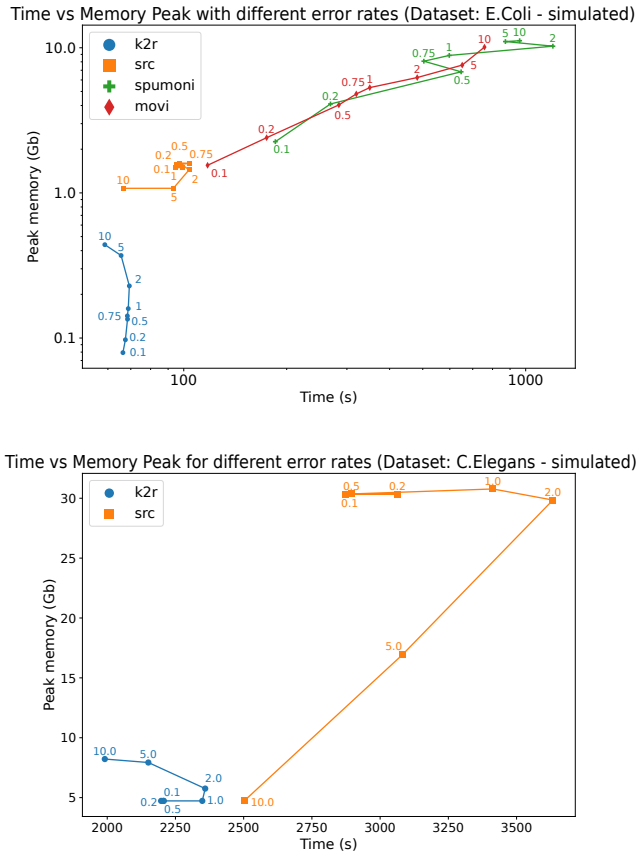

Fig. 5. Memory peak and CPU time used during the index construction according to the error rate (specified on the labels in percent) from simulated reads from *E. coli* (top) and *C. elegans* (bottom) genomes. In this experiment the reads were simulated from a base length of 10,000 with a 50X coverage.

further validated our findings with simulated reads from the *C. elegans* genome, yielding similar results consistent as depicted in Appendix Figure S6. In a scalability experiment involving the human datasets, we observed that indexing high-coverage human datasets is relatively cost-effective, requiring less than 10GB for the 390Gb ONT fasta file. Remarkably, the index for HiFi data is even smaller 207MB, underscoring how sequencing errors can dramatically affect index sizes.

Following the experimental plan, in a second experiment, we assess the error rate impact using simulated data as shown in Figure 9. As expected, a higher error rate generally means a larger index file, with the exception of SRC, where a higher error rate leads to a smaller number of kmers indexed and, therefore, a smaller index size. Apart from this effect, K2R is almost always an order of magnitude smaller than its competitors. Due to K2R's internal mechanisms, we anticipated that longer reads, for a given coverage, mechanically result in fewer reads and fewer distinct colors to index. To assess this impact, we conducted another simulated experiment showcasing the effect of read length on K2R, presented in Figure 10. Since it has almost no measurable impact on other tools, we focused on K2R's results; the plot with all tools is displayed in the Figure S9 Appendix. We observe that increasing the read length from 100 base pairs to 200,000 can indeed halve the K2R index size for a 1% error rate.

Queries

We now assess the query throughput of the different indexes. Assessing the query time for a single query would be challenging and to avoid accounting for overheads such as index loading, we chose to query batches of hundreds of sequences, each one thousand bases long, extracted from the *E. coli* reference genome to test "positive" queries. We report the elapsed time for growing batches in Figure 11. As we selected large enough batches, we observe almost no overhead, and all query times grow linearly with the input size. We find that K2R and Movi are equally fast and orders of magnitude faster than the two other tools, which require seconds to handle the batches. Memory-wise, the memory usage of all tools but SRC does not change according to the batch size. K2R is very frugal, using less than 200MB of RAM, while Spumoni uses more than 600MB and Movi more than 1GB. We also run "negative" queries constitued of random sequences of the same size and obtains almost identicals results (displayed in the Figure S7 and S8 of the Appendix).

In summary, our analysis leads us to three key conclusions. Firstly, the K2R index stands out as the most efficient for construction, matching the speed of SRC while significantly reducing memory consumption, in stark contrast to the resource-intensive nature of BWT-based indexes. Secondly, and

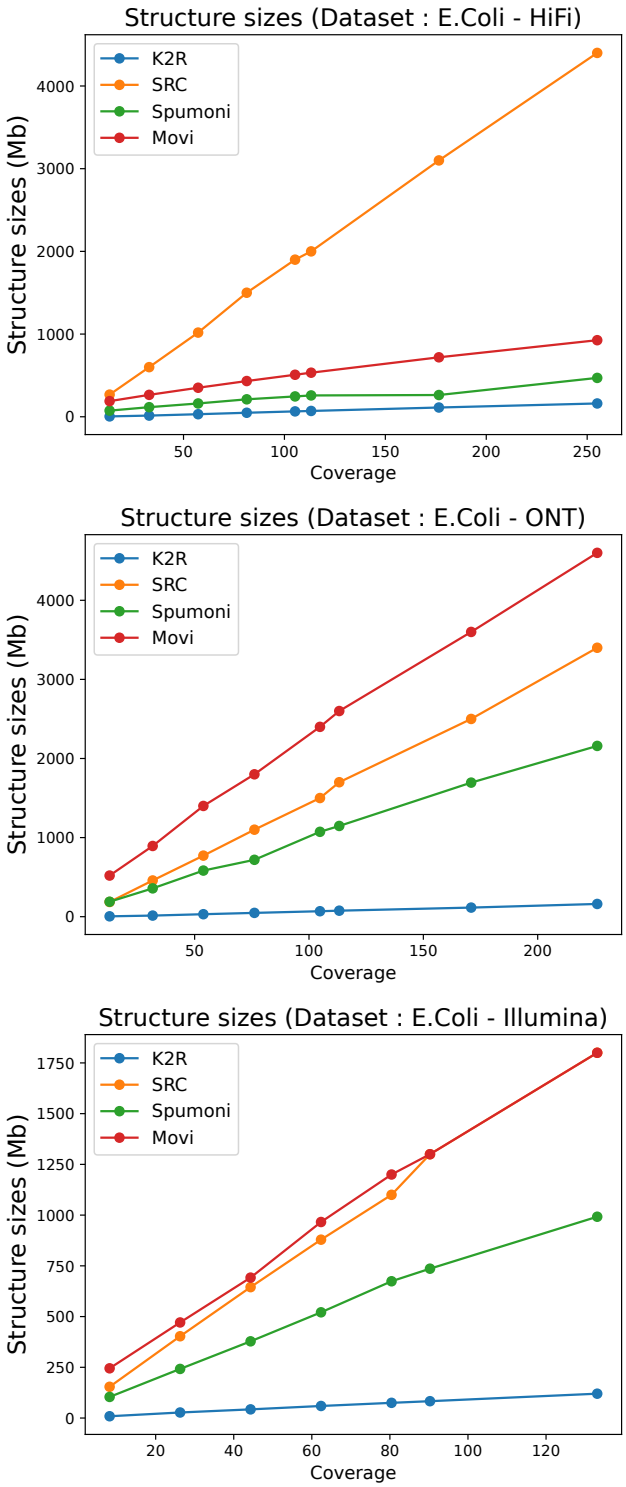

Fig. 6. Comparison of the index size, using HiFi data (top), ONT data (center) and Illumina data (bottom) reads from *E. coli* genome according to the input coverage.

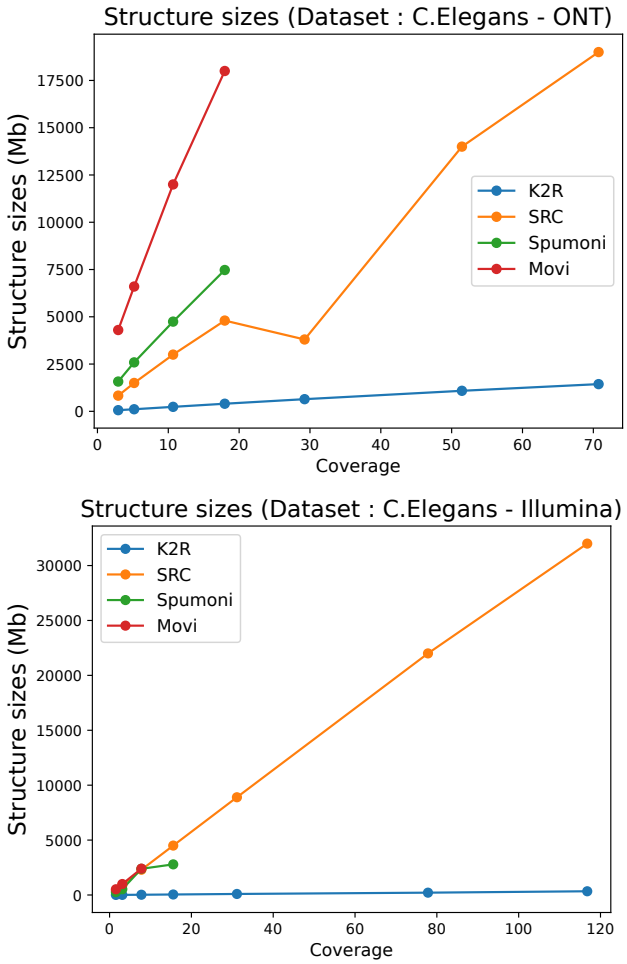

Fig. 7. Comparison of the index size, using ONT data (top) and Illumina data (bottom) reads from *C. elegans* genome according to the input coverage.

perhaps most critically, K2R indexes are markedly smaller than current leading tools, surpassing them by several orders of magnitude, independently of coverage and error rates. Thirdly, queries processed through K2R are exceptionally fast, a performance only paralleled by Movi, albeit Movi demands substantially more memory during query operations. These findings display the practical feasibility of employing Tinted de Bruijn graph methodologies at scale, highlighting the effectiveness of the presented approaches.

Conclusions and Future Work

There is a wide array of applications that could leverage the K2R index to achieve scalable read-level resolution, such as in metagenomics or transcriptomics quantification, assembly, or genotyping. Future work will aim to integrate and fine-tune it for these targeted applications.

Besides developing applications, there are multiple ways to improve our proposed implementation to make it more scalable and efficient. A trivial enhancement would be the use of static, memory-efficient associative structures, which could present an interesting trade-off for construction index efficiency. We propose to exploit two main characteristics of whole genome sequencing datasets: the existence of an underlying sequence and, therefore, overlapping reads, and the relatively uniform redundancy and error rates. To significantly improve performance beyond exploiting superkmers, a static

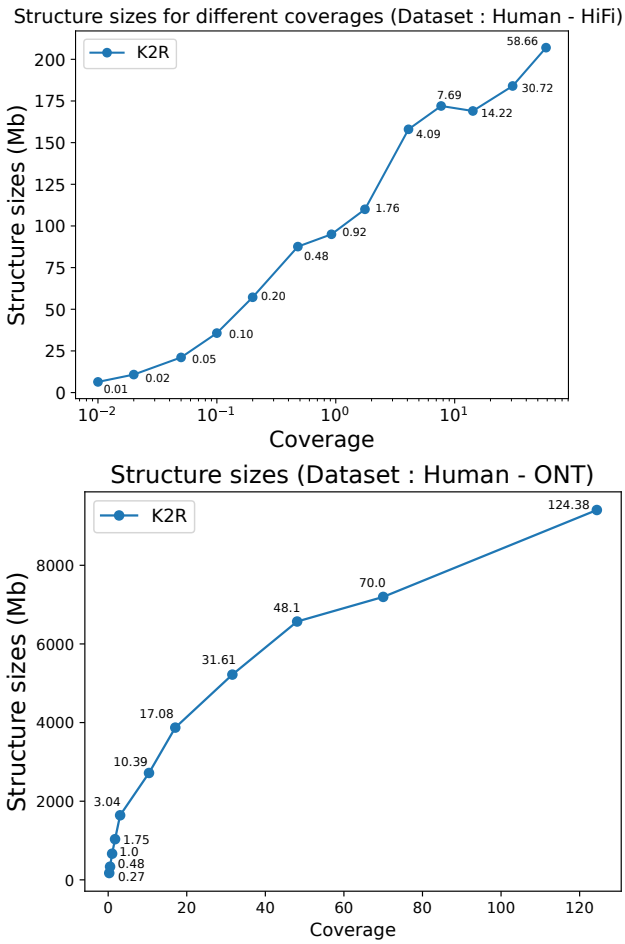

Fig. 8. Index size used during the index construction of a HiFi (top) and ONT (bottom) datasets from the *Human* genome according to the input coverage (also specified on the labels). The first graph is in logarithmic scale for readability purposes.

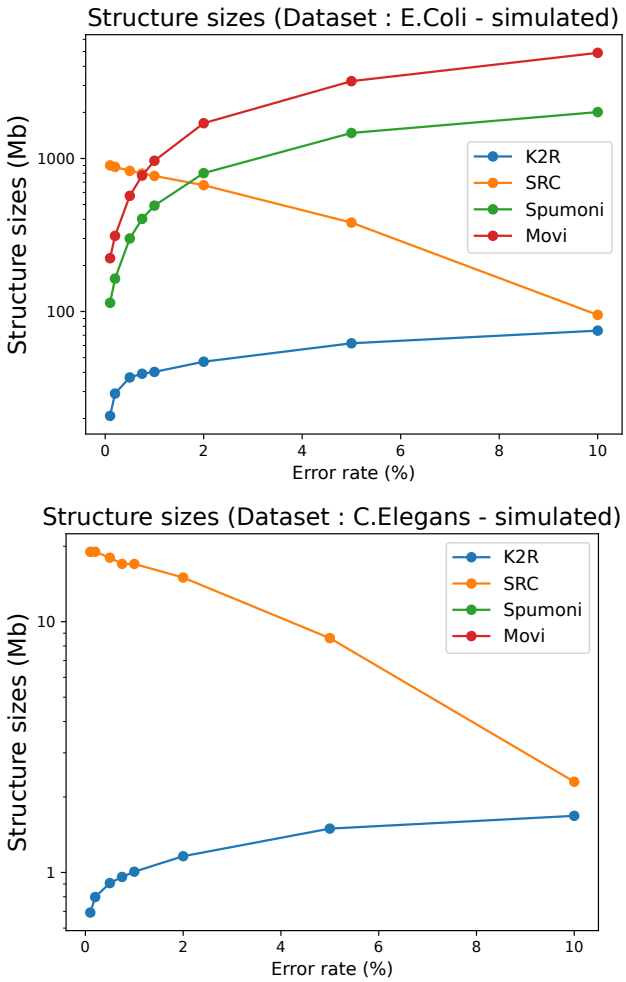

Fig. 9. Comparison of the index size according to the error rate from simulated reads from *E. coli* (top) and *C. elegans* (bottom) genomes. In this experiment the reads were simulated from a base length of 10,000 with a 50X coverage.

index could infer a good ordering (similar to the genome-induced ordering) based on some form of assembly. Using this ordering, we could exploit the fact that many successive kmers in this pseudo-genome would be associated with the same data, improving both query time and index size by reducing redundancy. Furthermore, when not identical, successive kmers often share very similar patterns (see Figure 2). Therefore, colors themselves could also be encoded relative to the previous colors, as we expect most modifications between two successive colors in the genome to involve the addition or removal of a read. Various degrees of such delta encoding could offer efficient time/memory trade-offs.

Another layer of optimization is that the list encoding could be highly compressible if related reads have close identifiers. By reordering the reads in the file to group similar/overlapping reads together, their shared kmers would be associated with highly compressed lists. Such read ordering could be based on clustering, overlap detection steps, or alignment to the aforementioned pseudo-genome. As a proof of concept, we compared the index size constructed from simulated reads in random order and ordered by their origin position in the genome. For a 200X coverage of *C. elegans* with a 1% error rate, the color index size reduced from 3.7GB to 1.3GB, a threefold improvement. With a 0.1% error rate, the color index size goes from 3.4GB to 251MB, an order of magnitude smaller. This display the interest of optimizing read ordering to facilitate their indexability or compression, a problem that start to regain interest for long reads datasets [17].

Even though the Minimizer Tinted de Bruijn graph is extremely lightweight and could fit many uses, an actual (kmer) Tinted de Bruijn graph could be interesting for applications where actual de Bruijn graph navigation is required or when a very low number of kmers are queried, and every false positive could be critical. Such an index could mimic the K2R representation and rely on efficient kmer indexes like SSHash [32].

Beyond these direct considerations, we believe that the Tinted de Bruijn graph structure is an intriguing representation of a sequencing dataset. Regular de Bruijn graphs are a robust representation of short-read datasets because they encapsulate most of their information in a lightweight and scalable manner. In this manuscript, we demonstrated that a lossy practical implementation is quite lightweight and scalable while retaining almost all the useful information from the original dataset. As

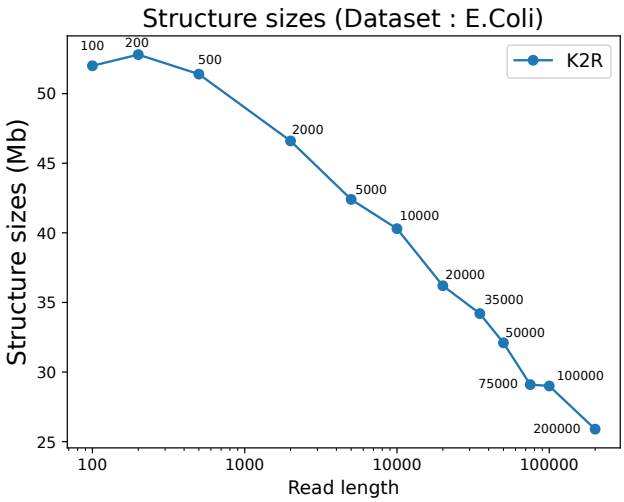

Fig. 10. Comparison of the index size according to the read length (also specified on the labels) from simulated reads from *E. coli* genome. In this experiment a 50X coverage of reads were simulated with an error rate of 1%.

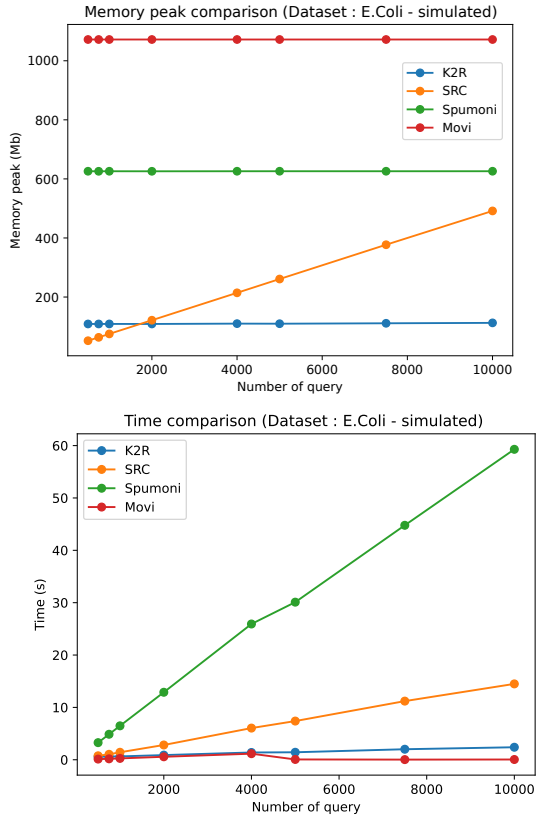

Fig. 11. Comparison of the memory peak (Top) and query time (Bottom) used during the query of different amount of sequences of size 1000 extracted without any errors from the reference genome. The queries are launched using indexes built from simulated reads with the followong caracteristics (coverage = 50X, error rate = 1%, read length = 10,000).

a result, while the de Bruijn graph has been deemed unsuitable for noisy long reads, we argue that the Tinted de Bruijn graph is the right representation for such datasets.

We also introduced the concept of the Reversible Tinted de Bruijn graph as a novel form of compressed full-text index adapted to sequencing data. Since we only need to manage duplicated kmers within a given read, we believe that the additional cost to allow a Tinted de Bruijn graph to be reversible will be quite low in most scenarios. The theoretical and practical study of such a structure will be the natural continuation of this work.

Competing interests

No competing interest is declared.

Acknowledgments

This work is funded the French National Research Agency AGATE ANR-21-CE45-0012.

References

1. O. Y. Ahmed, M. Rossi, T. Gagie, C. Boucher, and B. Langmead. Spumoni 2: improved classification using a pangenome index of minimizer digests. *Genome Biology*, 24(1):122, 2023.
2. J. N. Alanko, J. Vuohtoniemi, T. Mäklin, and S. J. Puglisi. Themisto: a scalable colored k-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes. *bioRxiv*, pages 2023–02, 2023.
3. F. Almodaresi, P. Pandey, M. Ferdman, R. Johnson, and R. Patro. An efficient, scalable and exact representation of high-dimensional color information enabled via de bruijn graph search. In *Research in Computational Molecular Biology: 23rd Annual International Conference, RECOMB 2019, Washington, DC, USA, May 5-8, 2019, Proceedings 23*, pages 1–18. Springer, 2019.
4. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.
5. A. Bankevich, A. V. Bzikadze, M. Kolmogorov, D. Antipov, and P. A. Pevzner. Multiplex de bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature biotechnology*, 40(7):1075–1081, 2022.
6. A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.
7. H. Bannai, T. Gagie, and I. Tomohiro. Refining the r-index. *Theoretical Computer Science*, 812:96–108, 2020.
8. G. Benoit, C. Lemaitre, D. Lavenier, E. Drezen, T. Dayris, R. Uricaru, and G. Rizk. Reference-free compression of high throughput sequencing data with a probabilistic de bruijn graph. *BMC bioinformatics*, 16(1):1–14, 2015.
9. N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter. Near-optimal probabilistic rna-seq quantification. *Nature biotechnology*, 34(5):525–527, 2016.
10. R. Chikhi, A. Limasset, S. Jackman, J. T. Simpson, and P. Medvedev. On the representation of de bruijn graphs. *Journal of Computational Biology*, 22(5):336–352, 2015.
11. R. Chikhi and P. Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014.
12. M. Heydari, G. Miclotte, Y. Van de Peer, and J. Fostier. Browniealigner: accurate alignment of illumina sequencing data to de bruijn graphs. *BMC bioinformatics*, 19(1):1–10, 2018.
13. T. Hu, N. Chitnis, D. Monos, and A. Dinh. Next-generation sequencing technologies: An overview. *Human Immunology*, 82(11):801–811, 2021.
14. Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 44(2):226–232, 2012.
15. C. Jain, A. Dilthey, S. Koren, S. Aluru, and A. M. Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. In *International Conference on Research in Computational Molecular Biology*, pages 66–81. Springer, 2017.
16. K. Kim, Y. Jeong, Y. Lee, and S. Lee. Analysis of counting bloom filters used for count thresholding. *Electronics*, 8(7):779, 2019.
17. D. Lee and G. Song. Fastqcls: a fastq compressor for long-read sequencing via read reordering using a novel scoring model. *Bioinformatics*, 38(2):351–356, 2022.
18. H. Li. Fast construction of fm-index for long sequence reads. *Bioinformatics*, 30(22):3274–3275, 2014.
19. H. Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.
20. A. Limasset, J.-F. Flot, and P. Peterlongo. Toward perfect reads: self-correction of short reads via mapping on de bruijn graphs. *Bioinformatics*, 36(5):1374–1381, 2020.
21. A. Limasset, G. Rizk, R. Chikhi, and P. Peterlongo. Fast and scalable minimal perfect hashing for massive key sets. *arXiv preprint arXiv:1702.03154*, 2017.
22. B. Liu, H. Guo, M. Brudno, and Y. Wang. debga: read alignment with de bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, 2016.
23. C. A. Lyman, M. S. Fujimoto, A. Suvorov, P. M. Bodily, Q. Snell, K. A. Crandall, S. M. Bybee, and M. J. Clement. Whole genome phylogenetic tree reconstruction using colored de bruijn graphs. In *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 260–265. IEEE, 2017.
24. C. Marchet, C. Boucher, S. J. Puglisi, P. Medvedev, M. Salson, and R. Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, 2021.
25. C. Marchet, M. Kerbiriou, and A. Limasset. Blight: efficient exact associative structure for k-mers. *Bioinformatics*, 37(18):2858–2865, 2021.
26. C. Marchet, L. Lecompte, A. Limasset, L. Bittner, and P. Peterlongo. A resource-frugal probabilistic dictionary and applications in bioinformatics. *Discrete Applied Mathematics*, 274:92–102, 2020.
27. C. Marchet and A. Limasset. Scalable sequence database search using partitioned aggregated bloom comb trees. *Bioinformatics*, 39(Supplement_1):i252–i259, 2023.
28. N. Nagarajan and M. Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology*, 16(7):897–908, 2009.
29. T. Nishimoto and Y. Tabei. Optimal-time queries on bwt-runs compressed indexes. *arXiv preprint arXiv:2006.05104*, 2020.
30. S. Nurk, S. Koren, A. Rhie, M. Rautiainen, A. V. Bzikadze, A. Mikheenko, M. R. Vollger, N. Altemose, L. Uralsky, A. Gershman, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022.
31. D. Pellow, L. Pu, B. Ekim, L. Kotlar, B. Berger, R. Shamir, and Y. Orenstein. Efficient minimizer orders for large values of k using minimum decycling sets. *Genome Research*, 33(7):1154–1161, 2023.
32. G. E. Pibiri. Sparse and skew hashing of k-mers. *Bioinformatics*, 38(Supplement_1):i185–i194, 2022.
33. G. E. Pibiri, Y. Shibuya, and A. Limasset. Locality-preserving minimal perfect hashing of k-mers. *Bioinformatics*, 39(Supplement_1):i534–i543, 2023.
34. L. P. Ramos, F. A. Louza, and G. P. Telles. Genome comparison on succinct colored de bruijn graphs. In *International Symposium on String Processing and Information Retrieval*, pages 165–177. Springer, 2022.
35. M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.

36. M. Rossi, M. Oliva, B. Langmead, T. Gagie, and C. Boucher. Moni: a pangenomic index for finding maximal exact matches. *Journal of Computational Biology*, 29(2):169–187, 2022.
37. B. Solomon and C. Kingsford. Improved search of large transcriptomic sequencing databases using split sequence bloom trees. *Journal of Computational Biology*, 25(7):755–765, 2018.
38. A. Trotman and J. Lin. In vacuo and in situ evaluation of simd codecs. In *Proceedings of the 21st Australasian Document Computing Symposium*, pages 1–8, 2016.
39. M. Wang, Y. Ye, and H. Tang. A de bruijn graph approach to the quantification of closely-related genomes in a microbial community. *Journal of Computational Biology*, 19(6):814–825, 2012.
40. M. Zakeri, N. K. Brown, O. Y. Ahmed, T. Gagie, and B. Langmead. Movi: a fast and cache-efficient full-text pangenome index. *bioRxiv*, pages 2023–11, 2023.
41. H. Zheng, C. Kingsford, and G. Marçais. Improved design and analysis of practical minimizers. *Bioinformatics*, 36(Supplement_1):i119–i127, 2020.