Indexing and Querying Sequences Using an Alignment-Free Method

Comparative Study of Unitigs and Simplitigs in SPSS Construction

Alexine, Julie Buset

December 16, 2024

1 Introduction

The exponential growth of high-throughput sequencing data has profoundly transformed bioinformatics, both in terms of analytical methods and computational infrastructures. Next-generation sequencing technologies produce immense volumes of data, thus requiring increasingly efficient approaches for storing, processing, and comparing DNA sequences. In this context, the use of k-mers—fixed-length sub-sequences of size k—has become central, offering a granular and flexible representation of genomic data. The effectiveness of this approach has been demonstrated in various domains, ranging from de genome assembly [1] to metagenomic classification [9] and transcriptomic profiling [3].

The construction of k-mer-centric data structures, such as de Bruijn graphs, enables the effective modeling of overlaps between sequences. These graphs exploit the inherent redundancy of the data, yet tend to become fragmented and grow significantly in size when dealing with large or highly diverse datasets. As a result, despite their advantages, these structures can be challenging to use in practice, leading to the development of more compact and efficient tools to address these limitations.

Several studies have addressed this challenge by proposing approaches aimed at reducing the size and complexity of these representations. Spectrum-Preserving String Sets (SPSS), which compact de Bruijn graphs by eliminating redundancy while preserving all present k-mers, represent a notable advance [6]. Simplitigs, a generalized form of unitigs, allow a more concise representation of genomic data by condensing the non-branching paths of the graph [2].

In this exploratory project, we aimed to examine the challenges of fragmentation and redundancy in de Bruijn graphs. We began by constructing traditional unitigs as a baseline representation. To test more compact and efficient methods, we focused on simplitigs, which reduce redundancy while preserving all k-mers. These representations were then indexed using an alignment-free FM-index, enabling rapid querying of shared k-mers between input sequences and the indexed dataset. By combining unitigs/simplitigs with the FM-index, we evaluate their potential in terms of graph compaction, index construction, and query performance.

De Bruijn Graph

The **de Bruijn graph** is a k-mer-centric structure used to model overlaps between genomic sequences. Each *k-mer*, a substring of fixed length k, is represented as a node, and directed edges connect nodes that share an overlap of $k-1$ nucleotides. This construction efficiently encodes relationships between k-mers and provides a scalable representation of genomic data.

Despite its efficiency, de Bruijn graphs face a major challenge when applied to large or highly diverse datasets: **fragmentation**. Fragmentation arises at branching nodes, where multiple paths diverge or converge due to factors such as genetic variations, repeats, or sequencing errors. These branches break otherwise contiguous paths into smaller segments, increasing the complexity of the graph. As a result, the graph size grows significantly, which complicates storage, traversal, and further analyses.

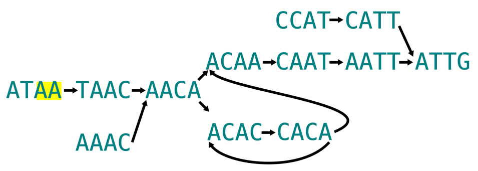

Figure 1: 1 k-mer set that is represented by the de Bruijn graphs simplified

In Figure 1, a de Bruijn graph constructed with $k = 4$ is shown. Nodes represent k-mers, and directed edges indicate overlaps of $k-1$ nucleotides. Simple paths, such as $ATAA \rightarrow TAAC \rightarrow AACA$, remain intact, while nodes like $ACAA$ introduce branching, fragmenting the graph into shorter paths. Additionally, cycles like those observed between $ACAC$ and $CACA$ highlight the structural redundancy commonly encountered in real-world datasets.

The fragmentation and complexity of de Bruijn graphs become particularly problematic when analyzing datasets with high levels of polymorphism or redundancy, such as bacterial pangenomes or metagenomic samples. To address these challenges, various methods have been proposed to simplify the representation of de Bruijn graphs.

Spectrum-Preserving String Sets (SPSS)

Spectrum-preserving string sets (SPSS) have emerged as an essential abstraction to compactly and effectively capture the k-mers in sequencing datasets. Introduced by Rahman and Medvedev (2021) [6], an SPSS is defined as a set of strings where (1) all k-mers from the original dataset are preserved, and (2) each k-mer appears exactly once. This ensures both completeness and uniqueness, making SPSS invaluable for tasks such as genome indexing, alignment-free analyses, and de Bruijn graph-based assembly. [5] Unitigs are one of the most well-known representations within the SPSS framework. In the context of de Bruijn graphs, unitigs are defined as maximal non-branching paths, meaning they traverse consecutive k-mers in the graph until encountering a branching point or dead end. This property ensures that each k-mer is represented exactly once in the resulting structure, maintaining both completeness and uniqueness. Unitigs effectively capture the structure of the graph, preserving connectivity between k-mers. However, this approach can result in a high level of fragmentation, especially in datasets with errors, polymorphisms, or high diversity. Fragmentation increases the number of unitigs, leading to larger and more complex representations that are less efficient to store and process. To address these challenges, "simplitigs" have been introduced as a generalized form of unitigs. Unlike unitigs, simplitigs do not stop at branching points, allowing them to produce longer, more compact sequences while still preserving all k-mers. By relaxing the strict structural constraints of unitigs, simplitigs focus on reducing redundancy, making them highly efficient for storage and indexing applications. This generalized representation maintains the completeness of k-mers while significantly reducing the number of resulting strings compared to unitigs, offering a more compact and practical solution for large-scale genomic analyses.

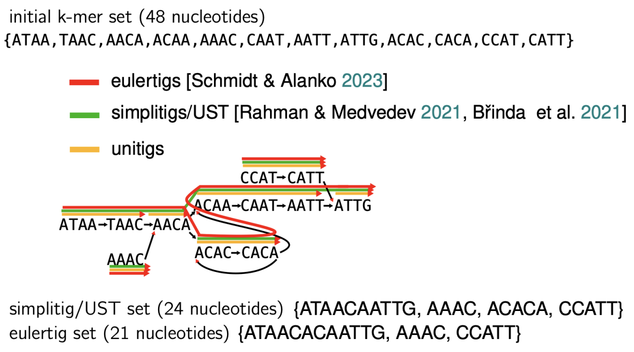
initial k-mer set (48 nucleotides)
{ATAA,TAAC,AACA,ACAA,AAAC,CAAT,AATT,ATTG,ACAC,CACA,CCAT,CATT}
eulertigs [Schmidt & Alanko 2023]
simplitigs/UST [Rahman & Medvedev 2021, Břinda et al. 2021]
unitigs
CCAT→CATT
ACAA→CAAT→AATT→ATTG
ATAA→TAAC→AACA
AAAC
ACAC→CACA
simplitig/UST set (24 nucleotides) {ATAACAATTG, AAAC, ACACA, CCATT}
eulertig set (21 nucleotides) {ATAACACAATTG, AAAC, CCATT}

Figure 2: Different examples of SPSS built from a same set of k-mers.

Figure 2 illustrates the differences between these SPSS representations, using the same set of k-mers to construct unitigs, simplitigs, and eulertigs. While the unitig representation adheres closely to the structure of the de Bruijn graph, the simplitig representation exploits k-mer contiguity to achieve a higher degree of compaction. Eulertigs [7], as shown, combine elements of both, further reducing redundancy while maintaining k-mer coverage. These representations highlight the trade-offs inherent in SPSS construction, depending on whether the primary objective is graph fidelity or data compression. The construction of SPSS varies depending on the specific use case, leading to several representations. Simplitigs, for example, are built by sequentially compacting k-mers, often using properties such as shared minimizers to group contiguous k-mers into longer strings. Recently, "eulertigs" were introduced as a novel representation, providing higher compression by merging paths in specific cases where unitigs or simplitigs would otherwise remain distinct. This approach strikes a balance between maintaining graph connectivity and achieving compression, offering practical benefits for downstream analyses.

The utility of SPSS lies in their ability to preserve the spectrum of k-mers while optimizing memory usage. For example, simplitigs compact a sequence dataset into fewer, longer strings without losing the k-mers, significantly reducing the size of the resulting data structure compared to naive k-mer enumeration. Unitigs, while less compact than simplitigs, are crucial in preserving the graph structure of the k-mers, making them indispensable for applications requiring a structural interpretation of sequence relationships. Meanwhile, eulertigs provide a compromise between these approaches by merging paths to achieve compression while retaining sufficient graph connectivity for downstream analyses.

The choice of SPSS representation is thus closely tied to the specific requirements of the task at hand. In applications where compactness is paramount, simplitigs or eulertigs may be favored. When the relationships between k-mers are critical, unitigs provide the necessary structural detail. Understanding these trade-offs is central to SPSS for large-scale genomic analyses, as the optimal representation depends on the balance between storage efficiency, computational speed, and the desired level of structural information.

2 Methods

This project implements a method for indexing and querying genomic sequences by the representation of k-mers and their storage in a compressed format. The approach is designed to work entirely without sequence alignment, focusing on comparing k-mer content to determine similarities between sequences.

To achieve this, the pipeline begins with the extraction of k-mers from the input genomic sequences. Each k-mer is transformed into its canonical form to ensure consistency and remove redundancies caused by reverse complements.

To handle sequencing errors, k-mers are filtered based on a solidity threshold, keeping only those that occur frequently enough to be considered reliable.

Next, the filtered k-mers are organized into a SPSS, constructed using a de Bruijn graph traversal. This step condenses the dataset by capturing all solid k-mers into a compact representation while eliminating duplicates. Finally, the SPSS is indexed using a Burrows-Wheeler Transform (BWT)-based FM-index. This index enables efficient querying by allowing rapid identification of shared k-mers between a query sequence and the indexed dataset, without the need for alignment.

This method is designed to scale with large genomic datasets, providing a accurate way to analyze sequencing data. Throughout our implementation process, we used **SnakeViz** to profile our scripts, identify performance bottlenecks, and optimize the most time-consuming functions to improve overall efficiency.

2.1 Processing Solid Canonical k-mers

Data Processing and Extraction

The first step involves extracting and counting canonical k-mers from the sequencing data. A canonical k-mer is defined as the lexicographically smallest sequence between a k-mer and its reverse complement. This canonicalization reduces redundancy and ensures that complementary sequences are treated equivalently, providing a uniform representation for downstream analyses.

The size of the k-mers, k is a **user-defined parameter** that controls the balance between sensitivity and specificity. Larger k values increase specificity by capturing longer, unique patterns but may reduce sensitivity due to fewer overlapping k-mers in shorter sequences. Smaller k values improve sensitivity but may lead to reduced specificity, as shorter k-mers are more likely to occur in unrelated sequences.

To count k-mers, the input FASTA file is processed sequentially. Each sequence is divided into overlapping substrings of length k, and the reverse complement of each k-mer is computed using a nucleotide translation table. The canonical form is determined by selecting the lexicographically smaller sequence between the k-mer and its reverse complement. A dictionary tracks the occurrences of each canonical k-mer, with keys representing k-mers and values representing their abundances. This frequency distribution forms the foundation for the subsequent filtering step.

Filtering of Solid k-mers

After k-mer counting, a filtering step identifies *solid k-mers*, defined as canonical k-mers that meet or exceed a *user-defined solidity threshold t*. This threshold is critical for reducing noise by eliminating low-frequency k-mers, which are likely artifacts of sequencing errors, while retaining biologically meaningful patterns.

It is essential that the solidity threshold t is chosen carefully in relation to k. When t is too large relative to k, the filtering process becomes overly restrictive, as larger k values inherently reduce the number of overlapping k-mers in the dataset. Thus, t must be selected to balance sensitivity and specificity, depending on the dataset's characteristics. Each canonical k-mer in the frequency dictionary is evaluated against t.

K-mers that satisfy $\text{count}(k\text{-mer}) \geq t$ are retained in the final set of solid canonical k-mers. The resulting set provides a compact and reliable representation of the dataset, suitable for efficient querying and indexing in downstream tasks.

To further refine the dataset, only the canonical representation of each k-mer is retained, ensuring that reverse complements are treated equivalently. This normalization step reduces redundancy by avoiding the inclusion of both orientations of the same k-mer. By retaining only canonical k-mers, the dataset is prepared later in simplitig generation, where the removal of redundant representations simplifies the construction of the Solid Prefix Sequence Set (SPSS) and improves its compactness for downstream applications.

2.2 SPSS Construction

The creation of the SPSS is based on the processing of solid canonical k-mers to generate either unitigs or simplitigs, depending on the chosen mode. In the **unitig** mode, k-mers are organized into maximal paths within the k-mer graph, preserving structural connectivity. In contrast, the **simplitig** mode merges overlapping k-mers to achieve higher compression. The resulting sequences are concatenated into a single string, separated by delimiters ('#') and terminated with a sentinel symbol ('\$'), forming a compact and traversal-ready representation suitable for downstream indexing and analyses.

Unitigs for querying

To represent genomic datasets compactly while maintaining the connectivity of k-mers, we construct *unitigs*. A unitig is a maximal contiguous sequence derived from the k-mer graph, where a k-mer's $k-1$-mer prefix leads to exactly one $k-1$-mer suffix, forming an unambiguous path. By traversing the graph along these paths, unitigs summarize the dataset while preserving its structural integrity.

The construction begins by building a directed k-mer graph, where each k-mer is represented as an edge connecting its $k-1$-mer prefix to its $k-1$-mer suffix. Unitigs are identified by traversing paths through nodes with exactly one outgoing edge (unambiguous nodes).

For each edge in the graph, a new unitig is initiated unless the edge has already been traversed. The unitig is extended iteratively in both directions until reaching a branching node or a dead end. During this traversal, visited edges are dynamically tracked to ensure each edge is incorporated into exactly one unitig, avoiding redundancies. Once the path can no longer be extended in either direction, the constructed unitig is considered maximal and added to the final set of unitigs.

This process continues until all edges in the k-mer graph are processed. The resulting unitigs compactly encode the connectivity information of the original k-mers, reducing redundancy while preserving the relationships between sequences. This structure is particularly advantageous for downstream analyses, as it minimizes storage requirements while retaining the essential features of the dataset.

Simplitigs for compression

While unitigs compactly represent the connectivity of k-mers, *simplitigs* extend this concept to further reduce redundancy and optimize dataset representation. A simplitig is a contiguous sequence formed by merging overlapping unitigs or k-mers into a single non-redundant sequence, ensuring that all k-mers are fully represented in a compressed form.

The construction begins with the set of solid canonical k-mers. A seed k-mer is randomly selected to initiate the simplitig, which is then iteratively extended in both forward and backward directions. Extensions are guided by matching the $k-1$-mer suffix or prefix of the current sequence with overlapping k-mers. Canonical k-mer representations are used throughout to eliminate reverse complement redundancies and maintain consistency.

At each step, possible nucleotide extensions (A, C, G, T) are explored, and the base that completes the next valid k-mer is appended to the sequence. The traversal continues until no further extensions are possible. The constructed simplitig is then considered maximal and added to the output. As k-mers are used in the extension process, they are dynamically removed from the dataset, preventing duplication and improving efficiency.

Simplitig construction continues until all k-mers are processed. Compared to unitigs, simplitigs achieve a higher degree of compression by merging paths that may not be directly connected in the k-mer graph. The resulting structure captures the full k-mer content while reducing redundancy, making it particularly suitable for applications like sequence comparison and storage. Despite being heuristic, this approach provides an efficient and compact representation of genomic datasets for downstream analyses.

To ensure the correctness of our simplitigs, we compared our results with those produced by ProphAsm, verifying that our implementation generated similar outputs in terms of k-mer coverage and representation.

2.3 Indexing and Querying

Efficient representation and searching of genomic sequences require advanced data structures that enable rapid substring detection while minimizing memory usage. In this section, we describe the construction of an FM-index for the SPSS and its application in substring queries.

Index Construction

Once the Simplified Perfect Sequence Set (SPSS) is generated, it is indexed using an FM-index, a space-efficient data structure that combines the Burrows-Wheeler Transform (BWT) and a suffix array to enable rapid substring searches. The suffix array is generated using the `tools_karkkainen_sanders` algorithm, which we reimplemented in Python 3 for compatibility and improved performance, as the original implementation was in Python 2. The suffix array provides a sorted list of all suffixes in the sequence, forming the backbone for efficient substring navigation.

Using the suffix array, the BWT is computed by rearranging the sequence such that each character corresponds to the character preceding a suffix in the sorted order. This transformation clusters similar substrings, making the structure highly compressible and well-suited for large genomic datasets. To support efficient query operations, auxiliary data structures are built alongside the BWT. A rank dictionary stores cumulative counts of each character at every position in the BWT, enabling rapid rank computations for substring matching. Additionally, a mapping dictionary (n) records the starting positions of each character in the sorted BWT, facilitating efficient access to character ranges during search operations. These components collectively allow the FM-index to provide both compact storage and fast query capabilities.

Querying

The FM-index enables efficient substring queries through a process known as backward search. This technique uses the properties of the BWT to iteratively refine the search range for a query substring by processing it in reverse order.

Backward search begins by initializing a search range $[l, r]$ spanning the entire BWT. For each character in the query, starting from the last character, the rank dictionary is used to compute the cumulative occurrences of the character up to specific positions in the BWT. The mapping dictionary (n) helps locate the precise range of the character within the BWT. This process iteratively narrows the search range until the query is either confirmed within the indexed sequence or determined to be absent if the search range becomes invalid $(l > r)$.

This efficient mechanism enables rapid substring detection without requiring storage of the full sequence, making the FM-index a crucial tool for alignment-free sequence analysis. The integration of the FM-index with the SPSS provides a compact and efficient framework for querying genomic datasets, allowing for reliable and high-speed pattern detection across large-scale data.

3 Results

3.1 Evaluation Methodology

The proposed method was evaluated across simulated sequencing datasets with varying coverage levels, error rates, and genome complexities to assess its efficiency, scalability, and robustness. The evaluation focuses on multiple performance metrics, including SPSS construction time, peak memory usage, and the impact of k-mer size k and solidity threshold t on the method's behavior.

The experiments were conducted using the following datasets:

- **reads_05x.fasta**: Simulated sequencing at 5× coverage, representing sparse k-mer conditions with low sequencing depth.
- **reads_30x.fasta**: Simulated sequencing at 30× coverage, corresponding to standard sequencing conditions.
- **reads_30x_poly.fasta**: Simulated sequencing at 30× coverage with mixed strains to evaluate the method's ability to handle genome heterogeneity.
- **reads_80x.fasta**: Simulated sequencing at 80× coverage to test performance and k-mer redundancy in high-coverage scenarios.

The evaluation analyzed the influence of k-mer size k and solidity threshold t on SPSS construction. The k-mer size was varied across $k = 21$, $k = 31$, and $k = 41$, allowing us to examine its effect on the compactness of the SPSS and the computational performance. The solidity threshold t, which filters low-abundance k-mers to reduce sequencing noise, was explored within the range $t = 2$ to $t = 5$. For datasets with lower coverage, increasing the solidity threshold results in significant k-mer loss, whereas higher coverage datasets retain more k-mers, allowing for robust SPSS construction.

To ensure the reproducibility and scalability of the experiments, the entire analysis pipeline was implemented using the **Snakemake** workflow management system. Snakemake was used to automate the construction of SPSS for various combinations of k and t, track performance metrics such as execution time and memory usage, and generate FM-index structures for querying. By automating the workflow, we ensured consistent and reproducible results across all datasets, with seamless scalability to accommodate large-scale experiments.

The following performance metrics were recorded to evaluate the method: (i) the elapsed time required to construct the SPSS for varying k-mer sizes and solidity thresholds, (ii) the peak memory usage during SPSS construction, and (iii) the size and count of the resulting SPSS. Additionally, we measured FM-index querying performance to validate the suitability of the generated SPSS for downstream applications, focusing on the time required to search for k-mers and the correctness of the results.

This methodological framework provides a rigorous evaluation of the proposed approach, highlighting its performance across datasets with varying sequencing depths and genome complexities, while ensuring reproducibility through the use of Snakemake.

3.2 Effect of k on SPSS Size

To assess the impact of k-mer size k on the SPSS size, we analyzed two datasets representing extremes in sequencing depth: *reads_05x* (low coverage) and *reads_80x* (high coverage). The smallest dataset (*reads_05x*) provides insight into the behavior of SPSS construction under conditions of sparse k-mers, where sequencing coverage is limited. The largest dataset (*reads_80x*) serves as a benchmark for high coverage, where redundancy between k-mers allows for extended overlaps and more efficient sequence compression. The solidity threshold t was fixed at 2 to filter low-abundance k-mers and ensure robust input for SPSS construction.

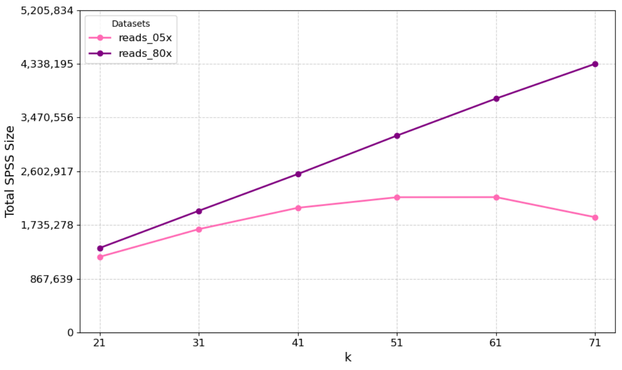

Figure 3: SPSS size for unitig construction.

For the unitig-based SPSS construction (Figure 3), the results demonstrate distinct behaviors depending on sequencing coverage. In the high coverage dataset (*reads_80x*), the SPSS size increases approximately linearly with k. This behavior arises from the high redundancy of k-mers, which enables the formation of extended unitigs even as k increases. In contrast, for the low coverage dataset (*reads_05x*), the SPSS size show a saturation effect beyond $k = 51$. As k increases, the sparsity of k-mers reduces the overlap between consecutive sequences, limiting the formation of longer unitigs and causing the SPSS size to plateau.

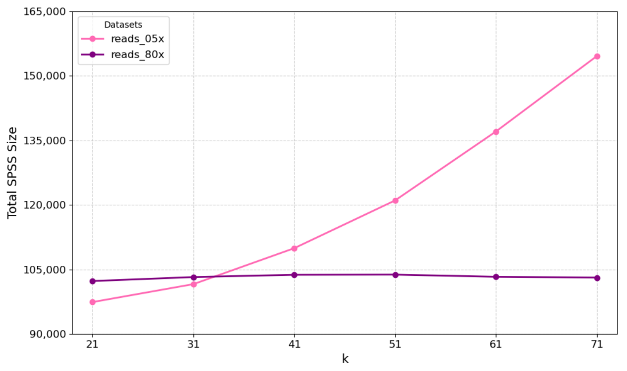

Figure 4: SPSS size for simplitig construction.

For the simplitig-based SPSS construction (Figure 4), the trends are markedly different. In the high coverage dataset (*reads_80x*), the SPSS size remains nearly constant across all values of k. This invariance reflects the compression efficiency of simplitigs, which merge overlapping k-mers into contiguous sequences while avoiding redundant representations. The abundance of k-mers in high coverage ensures sufficient overlaps, enabling consistent compression regardless of the k-mer size. In contrast, in the low coverage dataset (*reads_05x*), the SPSS size increases progressively with k. The reduced connectivity of k-mers at larger k-values limits the merging process, resulting in smaller simplitigs and a larger overall SPSS size.

These results demonstrate the contrasting effects of sequencing coverage on SPSS construction. By testing on the smallest and largest datasets, we highlight the extremes of SPSS behavior under low and high redundancy conditions. In high coverage datasets, simplitigs provide stable and efficient compression, while unitigs maintain linear growth. In low coverage datasets, however, increasing k exacerbates fragmentation, particularly in the unitig-based approach, leading to reduced compression efficiency. This analysis hilights the need to select k carefully: larger values of k increase specificity but can degrade SPSS performance under sparse sequencing coverage.

3.3 Impact of Solidity Threshold on SPSS Count and Size

We evaluated the effect of the solidity threshold t on SPSS size and count for simplitig-based construction across three datasets with distinct coverage and complexity levels: a low-coverage dataset (5x coverage), and two high-coverage datasets (30x coverage), one of which (30x_poly) comprises mixed strains. The solidity threshold t was varied from 2 to 5, and three k-mer sizes k (21, 31, and 41) were analyzed to assess their impact.

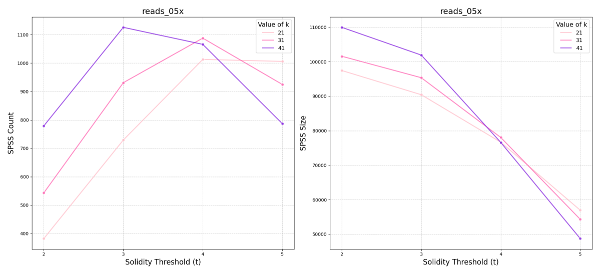

Figure 5: SPSS Count and Size as a Function of Solidity Threshold for the 05x Dataset

First, we analyzed the SPSS size and count for the same dataset to observe their variation with increasing solidity thresholds. For the 5x coverage dataset (Figure 5), the SPSS count was highest at $t = 2$, as low-frequency k-mers resulting from sequencing noise were retained. Despite this, the combination of $k = 21$ and $t = 2$ produced the most favorable results for this dataset. At these parameters, the SPSS count remains high, but the SPSS size is minimized relative to higher t values. This outcome reflects the fact that for low-coverage datasets, smaller k-mers (e.g., $k = 21$) better capture the underlying sequences, while a lower solidity threshold ($t = 2$) avoids excessive filtering that could eliminate biologically relevant k-mers. This conclusion is further supported by the results observed during the queries, which confirm that this combination preserves the majority of biologically relevant k-mers while minimizing noise and redundancy.

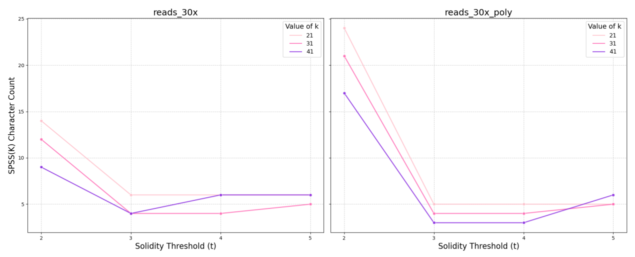

Figure 6: SPSS Count Across Different Solidity Thresholds for 30x and 30x poly Datasets

We then focused on analyzing the SPSS count in Figure 6, as for datasets with higher coverage, the solidity threshold t has a limited effect on SPSS size but significantly impacts the SPSS count. For the 30x coverage datasets, increasing t from 2 to 5 led to a gradual reduction in SPSS count, primarily due to the removal of low-frequency k-mers that are less supported by sequencing depth. At $t = 2$, the SPSS count was highest, reflecting the inclusion of more low-frequency k-mers resulting from sequencing noise or rare variants. As t increased to 3 and beyond, the SPSS count stabilized, indicating that most low-quality k-mers had been removed and that the remaining components were well-supported paths in the de Bruijn graph. The magnitude of the decrease in SPSS count was less pronounced compared to the 5x dataset, as higher sequencing coverage inherently provides more reliable k-mer frequencies, which reduces the presence of erroneous k-mers.

In the 30x_poly dataset, the SPSS count was consistently higher than in the single-strain 30x dataset across all thresholds. This increase is attributed to the presence of multiple strains, which introduces additional branching nodes and distinct paths into the de Bruijn graph, thereby increasing the number of SPSS components. Despite the higher complexity, the SPSS count followed a similar trend of reduction with increasing t. This behavior demonstrates that the solidity threshold effectively filters noise while retaining biologically meaningful diversity within the mixed strains. Importantly, the stabilization of SPSS count at higher thresholds indicates that the threshold primarily influences the number of branching paths while preserving the overall graph structure.

These results highlight the necessity of adapting the solidity threshold t and k-mer size k to the dataset's coverage and complexity. For the 5x dataset, the combination of $k = 21$ and $t = 2$ was shown to be optimal, as it retains biologically relevant k-mers while minimizing redundancy and error-driven components in the graph. In high-coverage datasets like 30x and 30x_poly, higher thresholds ($t \geq 3$) are effective at filtering low-frequency k-mers without significantly impacting graph completeness. Moreover, larger k-mers ($k = 31$) are advantageous for these datasets as they reduce graph ambiguity, particularly in complex cases such as the 30x_poly dataset, where the presence of mixed strains leads to more branching paths.

3.4 SPSS and Indexing Performance

The performance of SPSS construction and FMI indexing was evaluated for unitigs and simplitigs across our four datasets, each with varying coverage and complexity. The analysis examined three main aspects: the k-mer selection time, the SPSS construction time, and the FMI build time.

Dataset	Selecting k-mer (sec)	SPSS construction (sec)	Build FMI (sec)
reads_05x	0.26	0.14	4.58
reads_30x	1.69	0.14	6.08
reads_30x_poly	1.55	0.13	5.13
reads_80x	4.11	0.16	5.41

Table 1: Performance metrics for unitigs: k-mer selection, SPSS construction, and FMI build times across varying coverage datasets.

In the case of **unitigs**, the results highlight a significant discrepancy between SPSS construction efficiency and FMI build performance (Table 1). While the SPSS construction time remains consistently low across all datasets—ranging from 0.13 to 0.16 seconds—this is not reflected in the FMI build times, which increase considerably with higher coverage. For instance, in the dataset with a coverage of 30x, the FMI build time reaches **6.08 seconds**, indicating that the cumulative SPSS size for unitigs remains large. This observation suggests that the fragmentation of unitigs, caused by their strict adherence to non-branching paths in the de Bruijn graph, leads to a computational bottleneck during FMI construction. These inefficiencies become particularly pronounced in larger and more complex datasets, where graph branching is more prevalent.

Dataset	Selecting k-mer (sec)	SPSS construction (sec)	Build FMI (sec)
reads_05x	0.22	0.46	0.3
reads_30x	1.59	0.38	0.36
reads_30x_poly	1.52	0.45	0.45
reads_80x	4.15	0.4	0.39

Table 2: Performance metrics for simplitigs: k-mer selection, SPSS construction, and FMI build times across varying coverage datasets.

In contrast, **simplitigs** provide a more compact and computationally efficient representation (Table 2). By relaxing the non-branching constraint of unitigs, simplitigs allow paths to extend through branching nodes in the graph, thereby reducing the overall fragmentation. This design results in longer contiguous paths and a smaller cumulative SPSS size, which directly improves the FMI build time. For example, in the dataset with a coverage of 30x, the FMI build time for simplitigs is reduced to **0.36 seconds**, an order of magnitude lower than the time observed for unitigs. However, the improvement in FMI efficiency comes at the cost of increased SPSS construction time, which is consistently higher for simplitigs compared to unitigs. In the dataset with low coverage, the SPSS construction time for simplitigs is **0.46 seconds**, compared to **0.14 seconds** for unitigs. This increase reflects the additional computational effort required to identify and extend paths across branching nodes.

The observed results hilight the limitations of unitigs in handling datasets with high coverage or graph complexity. While unitigs achieve a minimal SPSS construction time, their fragmented representation significantly increases the cost of downstream FMI indexing. Simplitigs, on the other hand, provide a more scalable alternative by balancing the SPSS construction time with improved FMI build performance. This balance makes simplitigs a better choice for indexing large and complex datasets, where reducing cumulative SPSS size is critical for achieving computational efficiency.

3.5 Querying Performance

To evaluate the computational performance of querying sequences, we measured both the elapsed time and the peak memory usage for SPSS construction under varying values of k and t. The results, summarized in Table 3, highlight the differences in resource usage across datasets with distinct sequencing depths.

Dataset	k	t	Mode	Elapsed Time (mm:ss)	Peak Mem. (MB)
reads_05x	21	2	simplitig	00:0.19	34.89
			unitig	00:0.32	211.47
	31	2	simplitig	00:0.21	36.44
			unitig	00:0.45	296.39
reads_30x	21	2	simplitig	00:0.18	34.11
			unitig	00:0.34	231.77
	31	2	simplitig	00:0.20	33.73
			unitig	00:0.47	328.41
reads_30x_poly	31	3	simplitig	00:0.20	34.03
			unitig	00:0.46	344.55
	41	3	simplitig	00:0.23	34.59
			unitig	00:0.55	398.36
reads_80x	31	3	simplitig	00:0.22	34.48
			unitig	00:0.52	343.2
	41	3	simplitig	00:0.23	33.81
			unitig	00:0.56	414.47

Table 3: Querying performance under varying k-mer size (k) and solidity threshold (t): elapsed time and peak memory usage for unitigs and simplitigs.

The results show that **simplitigs consistently achieve lower memory usage and reduced construction time compared to unitigs**, regardless of the sequencing depth or complexity of the dataset. Across all tested configurations of k and t, the memory consumption for simplitigs remains remarkably stable, hovering around **34 MB**, even as the sequencing depth increases. In contrast, unitigs display a significant increase in resource usage, with peak memory values reaching up to **414.47 MB** for high-coverage datasets. This behavior reflects the fundamental limitation of unitigs, which are highly sensitive to graph fragmentation caused by branching nodes.

Elapsed time measurements further emphasize the efficiency of simplitigs. For lower values of k and t, simplitigs consistently construct the SPSS more quickly, with times as low as **19 seconds**. Unitigs, however, require considerably more time, with observed values up to **56 seconds** for higher coverage and larger k-mer sizes. As the sequencing depth increases, the computational burden associated with unitigs becomes disproportionately larger, while simplitigs maintain near-constant performance. This trend highlights the ability of simplitigs to scale efficiently, even in datasets with increased graph complexity or redundancy.

The performance gap is further amplified in more complex datasets, such as those combining multiple strains. Here, unitigs struggle to handle the additional branching introduced by strain diversity, resulting in inflated memory usage and construction times. Simplitigs, by contrast, remain efficient due to their relaxed construction constraints, which allow paths to extend through graph junctions, reducing the total number of SPSS components.

These observations collectively demonstrate that the superior efficiency of simplitigs stems from their ability to produce a more compact and less fragmented SPSS representation. While simplitigs mitigate fragmentation by relaxing non-branching constraints, **our implementation of unitigs** appears to exacerbate these limitations, particularly in high-coverage and complex datasets. The increased memory usage and construction times observed for unitigs suggest potential inefficiencies in their construction process, which may require further optimization to align with state-of-the-art approaches.

Conclusion

This study investigated methods for constructing, indexing, and querying Spectrum-Preserving String Sets (SPSS) to improve the efficiency of genomic data representation. While the advantages of simplitigs in terms of compactness, computational efficiency, and scalability were clearly demonstrated, the study also identified several areas for improvement in both implementation and methodology.

Our implementation of unitigs revealed inefficiencies, particularly in high-coverage and complex datasets. The increased memory consumption and FMI build times suggest potential optimization issues in the current implementation. Adopting more sophisticated methods for unitig construction, such as advanced traversal algorithms or heuristic approaches, could significantly improve performance.

We followed PEP8 standards using the flake8 tool and used a PEP8 analysis tool to correct formatting issues. We also implemented recommended docstrings for documentation, but they are somewhat verbose, which affects readability. Streamlining the documentation while maintaining clarity could improve the code's usability for future development.

The querying script used for SPSS revealed limitations in its output format. Current outputs are generated sequentially, which complicates downstream analyses. Optimizing the querying framework to produce consolidated or batch outputs would streamline workflows, making the tool more practical for large-scale data exploration.

The selection of k-mer size k and solidity threshold t was based on insights from existing literature. This provided effective results for SPSS construction and querying. However, further exploration of alternative parameters could uncover improved configurations for datasets with varying coverage, complexity, and sequencing errors.

Finally, while the chosen methods achieved promising results, exploring alternative representations could yield even better compression and query performance. Approaches such as **superstring constructions** [8], iterative SPSS decomposition [4], or integration with **colored de Bruijn graph techniques**, as proposed in recent studies, could provide deeper insights and utility of SPSS for genomic applications.

References

[1] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Com-*

putational Biology, 19(5):455–477, 2012. _eprint: https://doi.org/10.1089/cmb.2012.0021.

[2] Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *Genome Biology*, 22(1):96, December 2021.

[3] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, February 2012.

[4] Kazushi Kitaya and Tetsuo Shibuya. Compression of Multiple k-Mer Sets by Iterative SPSS Decomposition. *LIPIcs, Volume 201, WABI 2021*, 201:12:1–12:17, 2021. Artwork Size: 17 pages, 1270960 bytes ISBN: 9783959772006 Medium: application/pdf Publisher: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[5] Camille Marchet. Advancements in practical k-mer sets: essentials for the curious.

[6] Amatur Rahman and Paul Medvedev. Representation of k-mer sets using spectrum-preserving string sets.

[7] Sebastian Schmidt and Jarno N Alanko. Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time.

[8] Ondřej Sladký, Pavel Veselý, and Karel Břinda. FroM Superstring to Indexing: a space-efficient index for unconstrained k -mer sets using the Masked Burrows-Wheeler Transform (MBWT), November 2024.

[9] Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, March 2014.