Pro*k*rustean graph: An efficient data structure used to extract sequence information for arbitrary *k*-mer size ranges

Adam Park[1‡] and David Koslicki[1,2,3,‡]

[1] Department of Computer Science and Engineering, The Pennsylvania State University
[2] Department of Biology, The Pennsylvania State University
[3] Huck Institutes of the Life Sciences, The Pennsylvania State University
‡ Corresponding authors: akp6031@psu.edu, dmk333@psu.edu

Abstract. Despite the widespread adoption of k-mer-based methods in bioinformatics, a fundamental question persists: How to elucidate the structural transition of a k-mer set when the order switches to k'? Lacking a generalized answer, the problem has been considered application-specific, necessitating the effect of k to be studied empirically in each k-mer-based method, e.g. in areas of genome complexity, assembly, alignment, error corrections, etc.

The challenge lies in the dynamic information space of k-mers, which transforms entirely with varying values of k. We unravel the problem with a principle: substring co-occurrence provides a localized scope grouping k-mers of different lengths that behave similarly. This principle is embodied in a model, the Pro*k*rustean graph, which serves as a theoretical framework for analyzing the role of k in altering the distributions and properties of k-mers. Furthermore, the model can extract both overlap graphs and de Bruijn graphs of any order, indicating the potential emergence of a hybrid assembly graph.

Practically, a Pro*k*rustean graph is a space-efficient data structure capable of efficiently querying k-mer-related information across various k-mer sizes, with time complexity that is independent of the size range. We present a series of examples demonstrating its competitive performance in both time and space, comparable to purpose-built tools designed for single k values, such as KMC and GGCAT. A notable application involves counting maximal unitigs in de Bruijn graphs over the range of $k = 30 \ldots 50000$ using a large long read dataset, accomplishing this task in approximately 30 seconds. This is a task that poses significant challenges for current state-of-the-art tools.

Our construction algorithm of the Pro*k*rustean graph utilizes an (extended) Burrows-Wheeler transform as input, is easily parallelizable, and operates in $O(N)$ time where N is the cumulative input sequence length. We provide theoretical justification that the size of a Pro*k*rustean graph is $O(N)$, and in practice, is significantly smaller than N.

All code and algorithms are publicly accessible at: `https://github.com/KoslickiLab/prokrustean`.

1 Introduction

As sequencing data and the number of reference genomes increase each year, k-mer-based methods continue to gain popularity in bioinformatics. K-mers, overlapping substrings of length k, enable the design of computationally efficient methods, albeit with some loss of information. Understanding this loss is crucial in quantifying the effectiveness of a specific k in almost every k-mer-based method. Surprisingly, a generalized model elucidating the relationship between k values and the structural changes of the set of k-mers is lacking; rather, such analyses are usually considered application-specific, and k-mer-based methods have independently struggled with 'inferring the role of k from the behavioral quality of results,' both in theoretical works and practical data processing. However, such inferences are vulnerable without an understanding of k-mer dynamics, particularly due to the varying characteristics of genomic data. Therefore, this theoretical gap has forced applications to rely on comprehensive empirical analyses, even though a common intuition is clear: increasing k leads to more specificity but sparser similarity, while decreasing k results in more sensitivity but ambiguous similarity.

For example, in genome analysis, the number of distinct k-mers is used as a proxy for the complexity and structure of a genome [4, 20]. Increasingly large sizes of genome data requires experiments to select only a few k values [6], as state-of-the-art k-mer counting tools have advanced with fixed k configuration [17]. A recent study even suggests that counting k-mers across a range of k values provides additional insights into pangenomes [5]. Moreover, k-mer spectra analyses, which consider frequencies of k-mers, naturally suffer the similar problem of utilizing multiple k values or finding a "best" k in their experiments [2, 14, 24].

Similar issues are encountered in genome assembly, which often use k-mer de Bruijn graphs. Determining a k for assembly is particularly crucial in de novo approaches [13] but still relies on heuristic methods [7, 19].

Furthermore, creating multi-k de Bruijn graphs did not show a clear progress [3, 9, 23]. These "variable-order" approaches implemented a "k change" functionality to navigate the vast representations of de Bruijn graphs built with multiple k values. Such exhaustive approaches, sans additional theoretical insights, are unlikely to achieve a significant breakthrough due to inherent computational burden of storing or operating over the representations. Many other tasks addressing similar issues include genome alignment [16, 18, 21] and error correction [1].

Previous studies have predominantly adopted local/inductive approaches, where k-mers are compared against their immediate counterparts, $k+1$-mers. However, these analyses have limitations in generalization. Firstly, the basic components of structural changes in k-mers are often obscured by application-specific interpretations. Secondly, the entire information space is transformed with varying k values. In other words, the significance of a k-mer is determined within its own (k-mer) space, but this entire space evolves by k.

In contrast, a global approach would see k-mers as a subset of substrings, treating them as local representations within the larger contexts of substring similarity. Addressing this overarching information space leads to a critical shift. We have identified a localizable context, substring co-occurrence, which links two substrings that behave similarly regardless of their length and applications. In this scheme, a k-mer is implicitly grouped with a k'-mer, with the strong explanatory power that the presence of one implies the presence of the other.

We introduce the Pro$k$rustean graph, a novel sequence similarity representation, paired with an operation that rapidly extracts extensive k-mer information by enumerating k values. The principle of substring co-occurrence clarifies how the distributions, frequencies, locations, and extensions of k-mers evolve with k, providing a framework to quantify the information loss in k-mers. Despite its dense information content, which allows ready access to details of any-length substring and their neighbors, the Pro$k$rustean graph remains compact in both theoretical and practical aspects, and can be constructed from a Burrows-Wheeler transform in a space-efficient manner. Therefore, the Pro$k$rustean graph can support k-mer-based methods by enabling the analysis of k with optimized efficiency. Furthermore, with the role of k elucidated, its potential applications extend to multi-k functionalities.

In this work, we first introduce the concept of the Pro$k$rustean graph and detail its interface with k-mer contexts. We then establish a framework for "smoothly" exploring various k values by investigating key properties of the graph. We provide a variety of algorithms demonstrating the unique information structure stored in the Pro$k$rustean graph. Counting k-mers across a range of k values facilitates rapid computation of genome complexity. Similarly, counting unitigs of de Bruijn graphs demonstrates exceptional efficiency to approximate genome assembly quality, as shown in fig. 1. This second application implies that a Pro$k$rustean graph is a sort of the union of de Bruijn graphs of all order, where duplication in intersections is efficiently omitted.

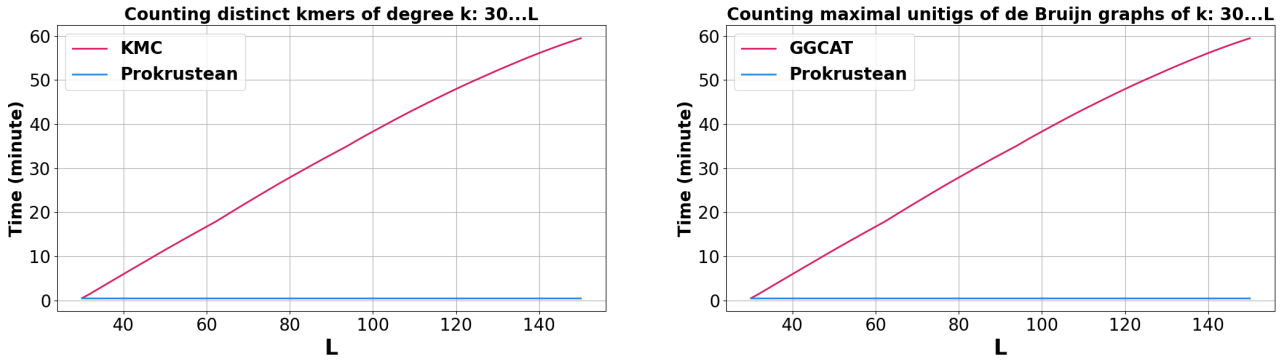

Fig. 1: The performance of Pro$k$rustean graphs for two representative functionalities were compared with state-of-the-arts: KMC [15] is a k-mer counting tool and GGCAT [8] constructs compacted de Bruijn graphs . Both were iterative called to extract k-mer/unitig counts of $k = 30 \ldots L$ of the sequences (4.5gb accumulated length, ERR3450203), while our algorithms (Section 4) using Pro$k$rustean graphs extract the whole information in a single run.

An interesting observation is that a Pro$k$rustean graph can efficiently extract both overlap graphs and de Bruijn graphs simultaneously. For over a decade, these two assembly graphs have developed in parallel without clear convergence. Overlap graphs account for all suffix-prefix overlaps between reads, whereas k-mer de Bruijn graphs, despite their efficiency, may miss or entangle extensions due to their fixed-overlap structure. An open problem in long-read assembly was called for adapting de Bruijn graphs to achieve the assembly quality of overlap graphs [22]. We propose that the Pro$k$rustean offers a promising approach for designing de Bruijn graphs with locally modified orders that prioritize overlap information.

2 Pro*k*rustean graph: Structuring similarities and accessing *k*-mers

2.1 Basic Notations

Let $\Sigma = \{A, C, T, G\}$ be our alphabet, $\Sigma^n = \{w_1 w_2 \cdots w_n | w_i \in \Sigma\}$ the set of all n-length strings, and $\Sigma^* = \bigcup_{n=1}^{\infty} \Sigma^n$ all finite length strings on the alphabet. We use $S \in \Sigma^*$ to represent a finite-length string, and $\mathcal{U} \subset \Sigma^*$ for a set of strings. Consider the following preliminary definitions:

Definition 1. – *A region in the string S is a triple (S, i, j) such that $1 \leq i \leq j \leq |S|$.*
- *The string of a region is the corresponding substring: $str(S, i, j) := S_i S_{i+1} ... S_j$.*
- *The size or length of a region is the length of its string: $|(S, i, j)| := j - i + 1$.*
- *An extension of a region (S, i, j) is a larger region (S, i', j') including (S, i, j):*

$$(S, i, j) \subsetneq (S, i', j') := i' \leq i \leq j \leq j' \text{ and } |(S, i, j)| < |(S, i', j')|.$$

- *The occurrences of S in $\mathcal{U}$ are those regions in strings of $\mathcal{U}$ corresponding to S:*

$$\mathrm{occ}_{\mathcal{U}}(S) := \{(S', i, j) \mid S' \in \mathcal{U},\ str(S', i, j) = S\}.$$

- *S is a maximal repeat in $\mathcal{U}$ if it occurs at least twice and more than any of its extensions: $|\mathrm{occ}_{\mathcal{U}}(S)| > 1$, and for all S' a superstring of S, $|\mathrm{occ}_{\mathcal{U}}(S)| > |\mathrm{occ}_{\mathcal{U}}(S')|$. I.e., given a fixed set $\mathcal{U}$, a maximal repeat S is one that occurs more than once in $\mathcal{U}$ (i.e. is a repeat), and any extension of which has a lower frequency in $\mathcal{U}$ than the original string S.*

2.2 Stratification

The key idea of *stratification* is to recursively capture locations of similarities in a compact, hierarchical representation. For instance, consider two sequences: ACCCT and GACCC. The simplest statement of their similarity is 'Both sequences have ACCC,' because all other similarities (e.g. CC or ACC) are covered by ACCC. When another sequence TCCCG is introduced, however, it quickly becomes challenging to make an agreement about the similarity. Consider two possible cases:

1. ACCC is in 2 sequences, and CCC is in all 3 sequences.
2. ACCC is in 2 sequences, and CCC is in 1 sequence TCCCG and 1 substring ACCC.

The first case uses 5 units of "information" (in terms of number of occurrences) but the second case uses 4 units while preserving the same meaning. This key concept as depicted in Figure 2.

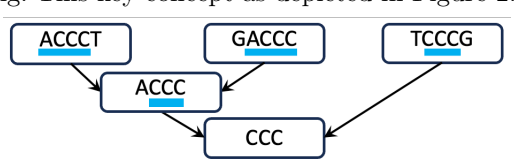

Fig. 2: Stratification. Given three strings ACCT, GACCC, and TCCCG, maximally matching subregions (indicated with blue underlining) are used to label terminal nodes of each edge. This process repeats hierarchically, hence the name *stratification*.

Definition 2. *A region (S, i, j) of a string S is stratifying in $\mathcal{U}$ if:*

$$|\mathrm{occ}_{\mathcal{U}}(str(S, i, j))| > |\mathrm{occ}_{\mathcal{U}}(S)| > 0 \tag{1}$$

and for each extension (S, i', j') of (S, i, j),

$$|\mathrm{occ}_{\mathcal{U}}(str(S, i', j'))| = |\mathrm{occ}_{\mathcal{U}}(S)|. \tag{2}$$

We call $str(S, i, j)$ a stratum in $\mathcal{U}$, and strata is the plural form.

In words, the string of a stratifying region appears more frequently in the string set $\mathcal{U}$ than its "parent" string S, but each extension appears as frequently as S. The blue regions in Figure 2 are straityfing and reveal the recursive structure. For example, we have that the region $(\text{ACCCT}, 1, 4)$ (i.e. ACCC) is stratifying in $\mathcal{U} = \{\text{ACCCT, GACCC, TCCCG}\}$. However, the region $(\text{ACCCT}, 2, 4)$ (i.e. CCC) is not stratifying in $\mathcal{U}$ because an extension $(\text{ACCCT}, 1, 4)$ (i.e. ACCC) appears 2 times in $\mathcal{U}$ which is more than the one-time occurrence of ACCCT. Then composing $\mathcal{U}' = \mathcal{U} \cup \{\text{ACCC}\}$, the region $(\text{ACCC}, 2, 4)$ (i.e. CCC) is stratifying in $\mathcal{U}'$.

Now, we collect strings identified from stratification. That is, starting with the initial set $\mathcal{U}$, we recursively capture stratifying regions and collect their strings. Each stratum obtained in this process is a maximal repeat. We later show indeed every maximal repeat is collected through this process.

Definition 3. *$\mathcal{S}_\mathcal{U}$ is the set strata of $\mathcal{U}$ if:*

1. *for each region (S, i, j) stratifying in $\mathcal{U}$ where $S \in \mathcal{U}$, $str(S, i, j) \in \mathcal{S}_\mathcal{U}$ and*
2. *for each region (S, i, j) stratifying in $\mathcal{U}$ where $S \in \mathcal{S}_\mathcal{U}$, $str(S, i, j) \in \mathcal{S}_\mathcal{U}$.*

Note: We often omit "in $\mathcal{U}$" and simply use "stratifying" when the context is clear.

In fig. 2, $\mathcal{U}$ = {ACCCT, GACCC, TCCCG} and $\mathcal{S}_\mathcal{U}$ = {ACCC, CCC}. Since the (ACCCT, 1, 4) is stratifying in $\mathcal{U}$, ACCC $\in \mathcal{S}_\mathcal{U}$, and since (ACCC, 2, 4) is stratifying in $\mathcal{U}$, CCC $\in \mathcal{S}_\mathcal{U}$.

2.3 Pro*k*rustean graph

A Pro*k*rustean graph is simply a graph representation of stratification. We emphasize that in the implementation of a Pro*k*rustean graph, the strings of $\mathcal{U}$ may not be stored; instead, labels indicating string lengths and region positions are usually sufficient.

Definition 4. *The Prokrustean graph of $\mathcal{U}$ is a directed multigraph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$ with sizes and intervals labeling vertices and edges, respectively:*

- $\mathcal{V}_\mathcal{U} : v_S \in \mathcal{V}_\mathcal{U}$ *if and only if* $S \in \mathcal{U} \cup \mathcal{S}_\mathcal{U}$
- $\mathcal{E}_\mathcal{U} : e_{S,i,j} \in \mathcal{E}_\mathcal{U}$ *if and only if* $(S, i, j) \in \bigcup_{S \in \mathcal{U} \cup \mathcal{S}_\mathcal{U}} \{(S, i, j) | (S, i, j) \text{ is stratifying}\}$
- $e_{S,i,j} \in \mathcal{E}_\mathcal{U}$ *directs from* v_S *to* $v_{str(S,i,j)}$
- *for* $v_S \in \mathcal{V}_\mathcal{U}$, $size(v_S) = |S|$
- *for* $e_{S,i,j} \in \mathcal{E}_\mathcal{U}$, $interval(e_{S,i,j}) = (i, j)$

2.4 Reflection

Reflection characterizes how the similarities identified in stratification are observed in terms of k-mers. Consider the example given in Figure 2, recalling that each label is partially covered with blue regions when similarities are found. For a given k-mer size, an *uncovered* region is a region of length at least k in the label of a node that is not a subregion of a blue region. (see the red underlining in Figure 3 where two different k-mer sizes are used to show uncovered regions).

Observe that k-mers are distinct to each other in these red regions. The key property is that every k-mer appears multiple times in blue/covered regions but only once in red/uncovered regions. In the graph on the left of Figure 3, for example, ACC is covered by two blue regions in the top-level nodes ACCCT and GACCC. In ACCC, however, ACC is included in a red region and does not appear again in any other red region. Similarly, CCC is covered by blue regions in all three top-level nodes and one middle node ACCC, but appears in the red region in CCC only once. The figure also reveals the rule of reflection behind increasing the k-mer size from $k = 3$ to $k = 4$. While the left graph ($k = 3$) preserves the CCC with a red region, bridging between two sequences ACCCT and GACCC and the other TCCCG, the right graph ($k = 4$) loses the connection. This phenomenon implies the disconnection event in genome assembly invoked by the transition of $k = 3$ to $k = 4$.

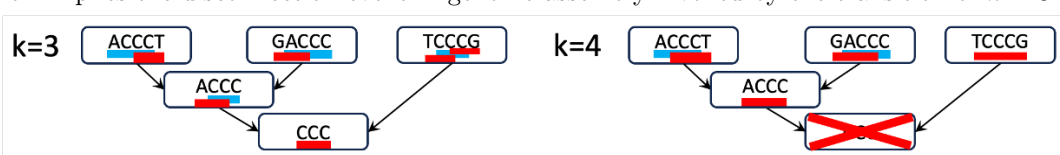

Fig. 3: Reflection. Depending on the size of k, the interpretation is done for $k = 3$ with the graph on the left, and for $k = 4$ with the graph on the right. Blue underlining depicts covered regions, and red underlining depicts uncovered regions. Reading the red regions is equivalent to enumerating k-mers each exactly once.

Definition 5. *A region (S, i, j) of a string S is k-reflecting in $\mathcal{U}$ if:*
for each k-mer K in $str(S, i, j)$,

$$|\operatorname{occ}_\mathcal{U}(K)| = |\operatorname{occ}_\mathcal{U}(S)| > 0, \tag{3}$$

and for each extension (S, i', j'), there exists a k-mer K in $str(S, i', j')$ that,

$$|\operatorname{occ}_\mathcal{U}(K)| > |\operatorname{occ}_\mathcal{U}(S)|. \tag{4}$$

We call $str(S, i, j)$ a k-reflectum in $\mathcal{U}$, and k-reflecta is the plural form.

In words, the string of a k-reflecting region maximally contains k-mers that co-occur with S. The co-occrrence has tremendous value applicable to most k-mer-based methods, as to be clarified in later section 3.

2.5 Computing reflection

Computing k-reflecting regions is trivial when stratifying regions are known. Assuming stratification has been computed, i.e. a Pro$k$rustean graph of $\mathcal{U}$ is given, we provide a simple procedure identifying k-reflecting regions with intuitive instructions in fig. 4. A formal algorithm is in Supplementary Section 1.1.

Definition 6. *For a string S, $reflect(S, k; \mathcal{U})$ is a procedure extracting k-reflecting regions in $\mathcal{U}$. When specifying the role of a Prokrustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$ in algorithms, we write $reflect(v_S \in \mathcal{V}_\mathcal{U}, k; \mathcal{E}_\mathcal{U})$.*

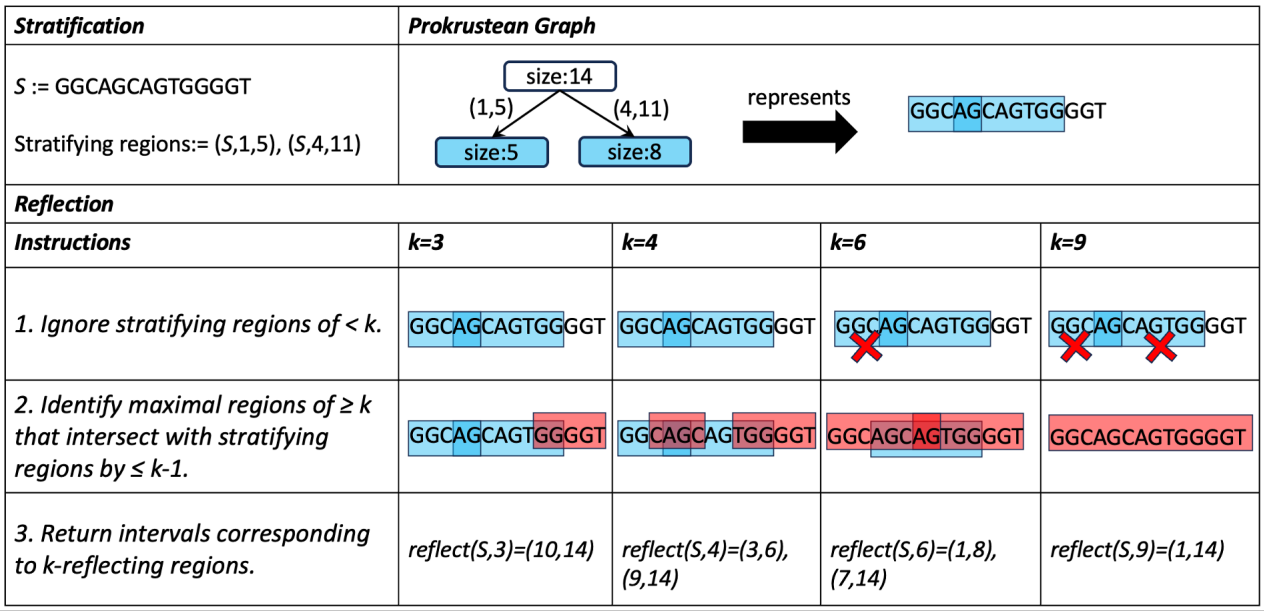

Fig. 4: Instructive description of $reflect(v_S \in \mathcal{V}_\mathcal{U}, k; \mathcal{E}_\mathcal{U})$. Decomposing $(S, 1, |S|)$ reveals k-reflectiong regions of S. $reflect$ captures "leftovers" of stratifying regions, but note that not only "gaps" between stratifying regions but intersections are considered. In an intersection between two stratifying regions, if a k-mer region is not included in both, e.g. the 4-reflecting region on CAGC, a k-reflecting region resides above the intersection. As k grows, reflecting regions dominate stratifying regions and are merged into one. E.g. the 9-reflecting region occupies the whole area.

2.6 Application: Identify all k-mers given any k

Before delving into the analysis of this model, we conclude this section with a basic application to facilitate familiarity with the model. Recall that the fig. 3 showed 3-mers and 4-mers are identified by red regions that correspond to 3-reflecta and 4-reflecta. Most importantly, k-mers in red regions do not duplicate.

The algorithm 1 enumerates $\mathcal{V}_\mathcal{U}$ and finds all k-reflecting regions from each vertex by the computation in fig. 4. The time complexity of algorithm 1 before returning k-mers is $O(|\mathcal{V}_\mathcal{U}| + |\mathcal{E}_\mathcal{U}|)$. The procedure $reflect$ in line 2 depends on outgoing edges of v_S, so the nested loop of line 1 and 2 is linear to the cardinality of the graph. Note that the whole running time is dominated by the output size in line 6.

Algorithm 1 Find all k-mers in $\mathcal{U}$

Input: Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, $k \in \mathbb{N}$
Output: k-mers in $\mathcal{U}$

1: **for** $v_S \in \mathcal{V}_\mathcal{U}$ **do**
2: **for** $(i, j) \in reflect(v_S, k; \mathcal{E}_\mathcal{U})$ **do**
3:   Collect (S, i, j)
4: **end for**
5: **end for**
6: Return k-mers in $S[i..j]$ for each collected region (S, i, j); requires an access to S.

3 Theory of k-mer variability across k ranges

A Pro$k$rustean graph is a similarity representation of sequence(s) that is compact and streamlined to explore between k-mer contexts of varying k values than any other models used in bioinformatics. We first describe the advantageous properties of stratification as a similarity representation, and then introduce a core concept - co-occurrence - that assigns a universal meaning to each k-mer within the system of substring similarity.

3.1 Stratification optimally encapsulates all substring similarity.

The following three theorems explain unique opportunities of a Pro$k$rustean graph. The first two theorems describe how maximal repeats and occurrences are exposed in the graph and the third theorem shows the size of the graph is still small. Note, all proofs omitted are found in Supplementary section 1.

Theorem 1 (Complete1). *A string R is a maximal repeat if and only if R is a stratum.*

It is trivial that a stratum is a maximal repeat, but the opposite is not immediate. Theorem 1 ensures exploring Pro$k$rustean graphs is equivalent to exploring sequences and maximal repeats. Then next theorem suggests it is not just strings of maximal repeats but their locations stored in the graph.

Theorem 2 (Complete2). *A region (S, i, j) in $S \in \mathcal{U}$ is an occurrence of a maximal repeat*

if and only if

a path from v_S to $v_{S,i,j}$ exists in the Prokrustean graph of $\mathcal{U}$ such that the edge intervals $(i_1, j_1), ..., (i_n, j_n)$ satisfies $i = \Sigma_{t=1}^{n} i_t - (n-1)$.

The occurrences of maximal repeats of $\mathcal{U}$ are stored in the topology of a Pro$k$rustean graph. Starting from a sequence vertex, nested stratifying regions represented by edges converge to a region of a maximal repeat represented by the terminal vertex. The next theorem shows the graph is near optimally compact.

Theorem 3 (Compact). *For $\mathcal{U}$ a set of sequences of total length N on the alphabet Σ,*

$$|\mathcal{S}_{\mathcal{U}}| < N, \ |\mathcal{E}_{\mathcal{U}}| \leq 2|\Sigma||\mathcal{S}_{\mathcal{U}}|, \text{ and hence } O(|\mathcal{V}_{\mathcal{U}}| + |\mathcal{E}_{\mathcal{U}}|) = O(N)$$

Thus, the cardinality of a Pro$k$rustean graph is $O(N)$. In practice, $|\mathcal{V}_{\mathcal{U}}| < 0.2N$ and $|\mathcal{E}_{\mathcal{U}}| < 2.5|\mathcal{V}_{\mathcal{U}}|$ were observed with short and long read data (see Section 4). Note, the well-known $O(N^2)$ amount of maximal repeat occurrences is burdensome and to our knowledge, no method has fully exploited them in linear space and time.

3.2 Reflection disjointly groups all substrings by co-occurrence

The major contribution of our work is to introduce co-occurrence as a key to analyze the similarity structure of k-mers. The intuition of 'one exists here if and only if the other exists here' is extremely useful to link substrings with different lengths having similar behaviors. We provide an intuitive principle and its proof to suggest a useful viewpoint to observe distributions of substring similarity.

Definition 7. *A string S <u>co-occurs in</u> a string S' in $\mathcal{U}$ if:*

$$S \text{ is a substring of } S' \text{ and } |occ_{\mathcal{U}}(S)| = |occ_{\mathcal{U}}(S')|.$$

Theorem 4 (Principle). *For any substring S such that $occ_{\mathcal{U}}(S) > 0$,*

$$S \text{ co-occurs in exactly one string in } \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}.$$

Proof. Assume the theorem does not hold: that S co-occurs in either no string or multiple strings in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$.

Consider the first case, when S co-occurs in no string in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$. Then S always occurs more than any string in which it occurs. So, S must be included in a stratifying region of those strings. Choose one such stratifying region and observe its string, a stratum. S occurs in the stratum, and by assumption its occurrence in the stratum is included in a stratifying region again. This process cannot continue forever because the lengths of the observed strata are strictly decreasing. Therefore, eventually, S is a substring of the last stratum when the process halts, but not occurring in a stratifying region. Clearly, S co-occurs in the last stratum, i.e. occurs in a $|S|$-reflecting region of the stratum, which is contradiction.

Now, consider the second case, when S co-occurs in at least two strings in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$. Consider two occurrences of S in each string, and extend the regions together either to the right or left by one at each step. Since the two strings are different, the strings of the regions should eventually diverge at least once, i.e. the number of occurrences decrease by extension before reaching both ends of both strings. This means S occurs more than at least one of those two superstrings, i.e. does not co-occur in at least one, which is contradiction. □

The above theorem intuitively shows that each substring, regardless of its length, can be considered as connected to exactly one sequence or maximal repeat. In the context of k-mers, it is clear from the proof that S co-occurring in a string is found in an $|S|$-reflecting region of the string, so a special case of the theorem is: *any k-mer occurs in exactly one k-reflectum in $\mathcal{U}$*. Therefore, we expand the principle to obtain two methodical corollaries. Here, we use $\bigsqcup$ to indicate a union of disjoint sets.

Corollary 1 (Fixed-k). *For a fixed k,*

$$\bigcup_{\forall S \in \mathcal{U}} \{k\text{-mer in } S\} = \bigsqcup_{\forall S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}} \bigsqcup_{\forall (S,i,j) \in reflect(S,k;\mathcal{U})} \{k\text{-mer in the } k\text{-reflectum } str(S,i,j)\}$$

This leads to the correctness of the algorithm in section 2.6. The right side is equivalent to (1) enumerating vertices $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$ of a Pro$k$rustean graph and (2) computing $reflect(S, k; \mathcal{U})$ to obtain k-reflecta and (3) splitting them into k-mers. The result coincides with the k-mers in $\mathcal{U}$ without duplication on the left. Next, moving the focus from k-mers to strings in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$ reveals another corollary supporting multi-k methodologies.

Corollary 2 (Varying-k). *For a fixed $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$,*

$$\text{The set of substrings that co-occur in } S = \bigsqcup_{\forall k \in 1 \ldots |S|} \bigsqcup_{\forall (S,i,j) \in reflect(S,k;\mathcal{U})} \{k\text{-mer in the } k\text{-reflectum } str(S,i,j)\}$$

This perspective treats each $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$ as a representative of a well-organized local scope. First, the representative strings in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$ do not share any substring in $\mathcal{U}$. Second, substrings under each representative inherit its properties such as frequencies, locations, and possible extensions. Lastly, with varying k, the composition of k-reflecting regions in S depends only on stratifying regions in S, i.e. outgoing edges of $v_S \in \mathcal{V}_{\mathcal{U}}$, and hence forms a local scope of co-occurring k-mers.

3.3 Co-occurrence reveals behaviors of k-mers via their representatives

The framework of k-mers co-occurring in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$ has been derived: Fix $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$, then vary k and observe how the distributions of k-reflecting regions change. This provides ample information about co-occurring k-mers that are interpreted in conjunction with S, and the analysis easily extends to the whole space. Such information includes the locations, frequencies, counts, and extensions of k-mers.

Locations of co-ocurring k-mers: These are static information per each $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$. Trivially, k-mers co-occurring in S always appear together in their relative positions in S. The information is stored in the Pro$k$rustean graph as paths as described in theorem 2. In practice, however, since computing maximal repeats is a part of the construction algorithm of a Pro$k$rustean graph, so gathering locations during the construction is usually more efficient. With the counts of k-mers co-occurring in S being computed, by sequentially increasing k, the progress of k-mers' presence and absence in sequences can be traced quickly.

Frequencies of co-ocurring k-mers: These are static sub-information of locations. Trivially again, k-mers co-occurring in $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$ share the same frequency which is the number of their locations. Again, computing frequencies of maximal repeats is efficient during the construction of the Pro$k$rustean graph. Consequently, changes of k-spectra can be analyzed quickly along with the counts of co-occurring k-mers.

Counts of co-ocurring k-mers: These are dynamic information following the changes of distributions of k-reflecting regions in S. As depicted in fig. 5, there are two scenarios affecting the numbers of co-occurring k-mers in S while changing k, when the composition of k-reflecting regions does not change, e.g. at the progress $k \leftarrow k+1$ no stratifying region has length k. This observation allows a rapid implementation of counting k-mers of all k's introduced in the next section section 4.2, taking $O(|\mathcal{V}_{\mathcal{U}}| + |\mathcal{E}_{\mathcal{U}}|)$ time.

(Right) extensions of co-ocurring k-mers: These are dynamic information following the k-reflecting region at the right end of S. As depicted in fig. 5, there are two scenarios deciding the extensions of each co-occurring k-mer in S, and the factor affecting the topology of assembly graphs (de Bruijn graphs) is when such k-mer is a suffix of S. The suffix k-mer and S share the same extensions, and the extensions of strata ($\mathcal{S}_{\mathcal{U}}$) can be collected during the construction of the Pro$k$rustean graph. Therefore, identifying the k values placing a k-reflecting region at the right (and left) end of S is enough to analyze the topology of de Bruijn graphs.

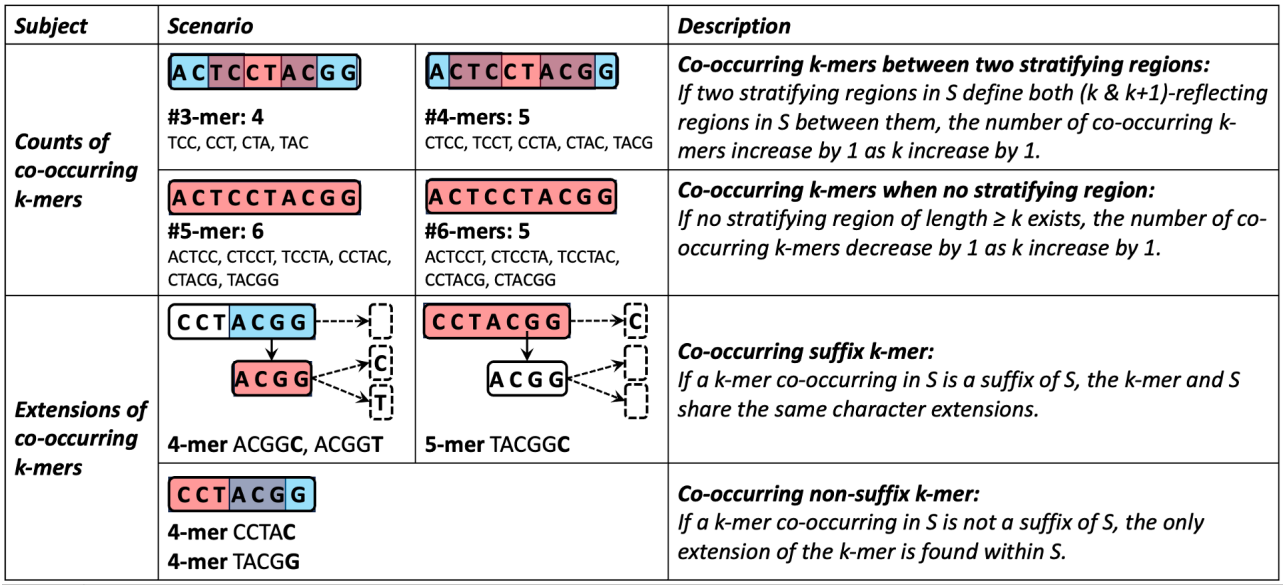

Fig. 5: Analysis of k-reflecting regions when k increases. Both counts and right extensions of co-occurring k-mers in a string are analyzed under the condition of increasing k. First, there are two cases in counting k-reflecting regions, assuming the composition of regions did not change by increasing k. The last unstated case is when a k-reflecting reaches only one end. Then increasing k does not affect the number of co-occurring k-mers. Extensions of co-occurring k-mers give interesting implications to genome assembly and alignment. Each maximal repeat influences the topology of a de Bruijn graph if and only if a k-mer is a co-occurring suffix of it.

4 Applications

We introduce the Pro*k*rustean graph construction and three applications designed to demonstrate the k-exploring capacity of the Pro*k*rustean graph. Most details are in the supplementary sections and implementation code. Five read datasets are used in this section: two short read datasets SRR20044276 (a microbial isolate sample) & ERR3450203 (a WGS metagenomic sample), a long read metagenome SRR18495451, and two human transcriptome short read datasets SRR21862404 & SRR7130905. These were randomly selected to represent variety in size, sequencing technology, and biological origin.

4.1 Pro*k*rustean graph construction

Our implementation takes $O(N)$ time and space where N denotes the cumulative sequence length of $\mathcal{U}$. While any of suffix tree, suffix array, and extended Borrow-Willer transform (ebwt) can be used for the purpose, we chose ebwt for its space-efficiency. Table 1 shows the resource requirements for each case. Note that $Lmin$ stands for the minimum length of stratum to be identified and so k-mers are not considered below this value Increasing $Lmin$ significantly decreases the output size, allowing application-specific adjustments.

dataset	#reads	#bp(N)	Lmin	time	memory	output	thread	$\lvert\mathcal{V}_\mathcal{U}\rvert$ & $\lvert\mathcal{E}_\mathcal{U}\rvert$
SRR20044276(metagenomic)	0.94M	72M	20	13s	200MB	40MB	8	2.7M/5.4M
ERR3450203(metagenomic)	55M	4.5B	30	28min	11GB	2.7GB	8	221M/407M
SRR18495451(metagenomic/long)	0.83M	11.3B	30	90min	43GB	14GB	8	890M/1.7B
SRR7130905 (human/rna-seq)	45M	6.8B	30	39min	18GB	4GB	8	331M/611M
SRR21862404 (human/rna-seq)	336M	22.3B	30	99min	37GB	5.9GB	8	667M/838M

Table 1: Pro*k*rustean graph constructions. Symbols M, B, MB, and GB mean millions, billions, megabytes, and gigabytes. The column #bp is the cumulative length of reads. The choice of $Lmin$ makes outputs vary in lower k sizes and we chose $Lmin = 30$ for most cases which is common in genome assembly practices. Observe that the outputs are at least comparable to the input size and $\mathcal{V}_\mathcal{U}$ and $\mathcal{E}_\mathcal{U}$ are smaller than the sequence length.

4.2 Application: Counting distinct *k*-mers for all *k* values

The number of distinct k-mers is an important metric for measuring the complexity of a genome, and is also relevant for k-spectra analysis by appending the frequencies of the k-mers in the context. Fixing $S \in \mathcal{U} \cup \mathcal{S}_\mathcal{U}$, and referring to the analysis in fig. 5, we constructed an invariant for the counts of co-occurring k-mers in S. Subsequently, we developed an algorithm (algorithm S3) based on this invariant that runs in $O(|\mathcal{V}_\mathcal{U}| + |\mathcal{E}_\mathcal{U}|)$

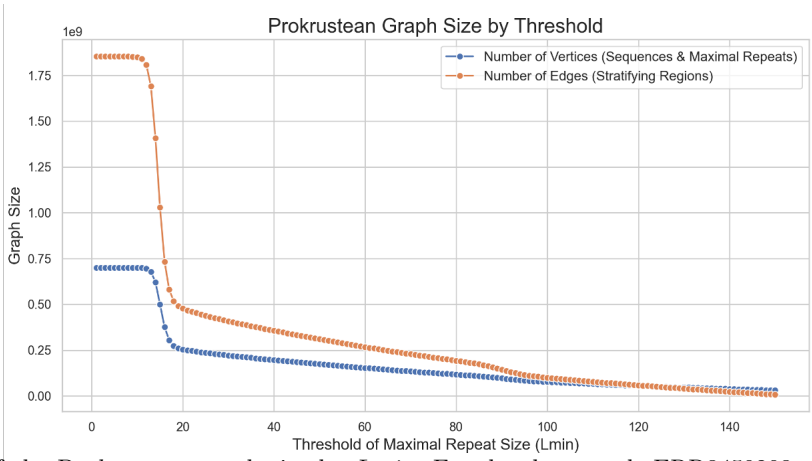

Fig. 6: An example of the Pro*k*rustean graph size by *Lmin*. For the short reads ERR3450203, the number of vertices and edges in the Pro*k*rustean graph were plotted as *Lmin* increases. On this data, the size of the graph drops around $Lmin = [11...19]$, meaning the vertices and edges are dense when *Lmin* is under 11 and the decrease of low-length maximal repeats are trivial. Another drop is at $Lmin = [81...89]$. Since vertices represent sequences too, the graph eventually becomes disconnected, and the number of edges falls below that of the vertices.

time. We compare to KMC [15], a fast, fixed k, k-mer counter. Since the time complexity is independent of the range of k, the results of experiments show a widening performance gap compared to state-of-the-art tools as the range gets larger, because other tools operate with a fixed k.

Dataset	k	Thread	KMC		Prokrustean Graph			
			time	memory	time (overall)	time (compute)	time (load/save)	memory
SRR20044276	1...150	8	5m	837mb	**0.36s**	0.06s	0.3s	80mb
ERR3450203	30...150	8	60m	12gb	**30s**	5s	25s	5.8gb
SRR18495451	30...50000	8	days	-	**88s**	13s	65s	24gb
SRR7130905	30...150	8	66m	12gb	**46s**	6s	40s	8.5gb
SRR21862404	30...150	8	112m	13gb	**74s**	8s	66s	13.8gb

Table 2: Counting distinct k-mers with a range of k. The time taken by the Pro*k*rustean graph, especially computing time after the graph is loaded, is not affected by the range size of k. KMC was called iteratively for each k, so it cannot avoid the steady increase of total amount of time, with inability to complete on the long read data in reasonable time.

Sequence Cardinality δ: Above result can be directly applied to computing Delta, a metric recently suggested to measure the sequence growth and similarity [5]. It is defined as $\delta(S) := \max_k (d_k(S)/k)$ where $d_k(S)$ is the number of distinct k-mers in S. Therefore, a Pro*k*rustean graph can be used to compute the number without iteratively counting k-mers for each k.

4.3 Application: Counting maximal unitigs of de Bruijn graphs for all *k* values

In genome assembly, counting simple paths in assembly graphs plays a key role to understand their topology. Maximal unitigs, non-extendable simple paths in de Bruijn graphs, are useful for designing metrics of assembly quality. The task of counting maximal unitigs for all k values takes $O(|\mathcal{V}_{\mathcal{U}}|)$ time when using a Pro*k*rustean graph (section 2.2). We compare with GGCAT [8], an efficient compacted de Bruijn graph generating algorithm.

Dataset	k	Thread	GGCAT		Prokrustean Graph			
			time	memory	time (overall)	time (compute)	time (load/save)	memory
SRR20044276	20..150	8	13m	310mb	**0.36s**	0.05s	0.31s	70mb
ERR3450203	30..150	8	98m	1.1gb	**31s**	5s	26s	5.8gb
SRR18495451	30..50000	8	days	-	**127s**	18s	109s	26gb
SRR7130905	30...150	8	126m	1.4gb	**46s**	6.7s	39s	8.5gb
SRR21862404	30...150	8	168m	0.8gb	**76s**	9s	67s	13.8gb

Table 3: Counting maximal unitigs with a range of k. The Pro*k*rustean approach is orders of magnitude faster than GGCAT, which is not designed for this task and was called each k separately to compute de Bruijn graphs.

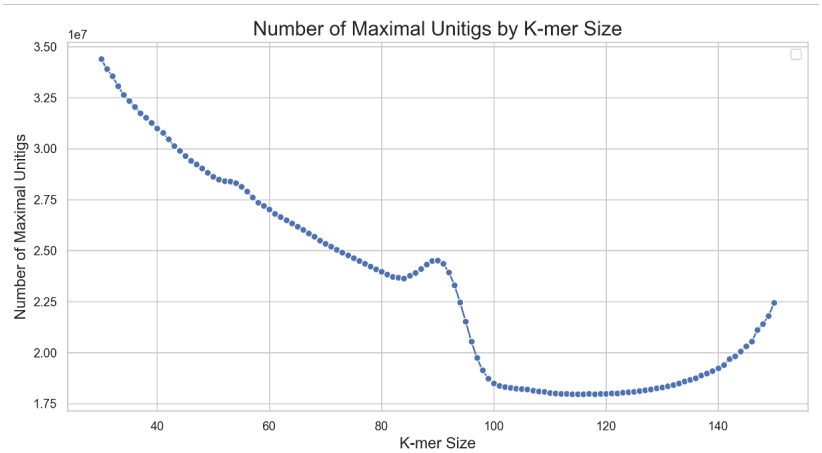

Fig. 7: An example of the number of maximal unitigs by k. In the de Bruijn graphs of ERR3450203, the number of maximal unitigs reveals complex scenarios. Maximal unitigs increase at both disconnections and intricate connections within the graph. The rise around $k = 90$ corresponds to the sudden decrease in maximal repeats as observed in fig. 6. Subsequently, further disconnections may eliminate branches, resulting in a period of stable short contigs, before the graph becomes completely disconnected.

4.4 Application: locally customizable assembly graphs

The previous example indicates that a Pro*k*rustean graph contains the union of de Bruijn graphs of all orders. Therefore, another potential contribution of a Pro*k*rustean graph in genome assembly is in advancing the so-called variable-order scheme by building a de Bruijn graph-like structure absorbing overlap information as much as possible. The algorithm 2 extracts an overlap graph from a Pro*k*rustean graph. Each stratum is the overlap information. Meanwhile, locations of strata in reads are "color" information in genome assembly contexts especially in de Bruijn graphs.

Algorithm 2 Extract the overlap graph of $\mathcal{U}$

Input: Pro*k*rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, $L \in \mathbb{N}$
Output: The overlap graph of $\mathcal{U}$ of the threshold L (least overlap length)

1: Make $(V, E) = (\{v_S \in \mathcal{V}_\mathcal{U} | S \in \mathcal{U}\}, \varnothing)$.
2: **for** $v_S \in \mathcal{V}_\mathcal{U}$ such that $S \in \mathcal{S}_\mathcal{U}$ and $size(v_S) \geq L$ **do**
3: Set $V_l = \{v_{S_l} \in V | S$ is a prefix of $S_l\}$, $V_r = \{v_{S_r} \in V | S$ is a suffix of $S_r\}$.
4: $E = E \cup V_l \times V_r$ where the new edges are labeled $size(v_S)$.
5: **end for**
6: Return (V, E).

As a proof of concept of the relatively complicated algorithm of extracting a de Bruijn graph from a Pro*k*rustean graph, we implemented the transformation. The k values are assigned per each $S \in \mathcal{U} \cup \mathcal{S}_\mathcal{U}$, so we could locally modify the order if desired.

	dataset	CPU-time	memory	thread	time (compute)	time (load/save)	k
Prokrustean	ERR3450203 (short/metagenome)	10m	20gb	1	8m	2m	30
	SRR21862404 (long/metagenome)	51m	67gb	1	47m	4m	30
GGCAT	ERR3450203 (short/metagenome)	9m	1.1gb	1	9m		30
	SRR21862404 (long/metagenome)	9m	1.1gb	1	9m		30

Table 4: **Comparison with fixed k:** De Bruijn graphs of fixed k values are computed. The CPU-time was measured as (threads*total time) because the prototype of the Pro*k*rustean approach is single threaded. The performance is slower in CPU-time than the purpose-built, optimized GGCAT. However, the Pro*k*rustean graph assigns each k at each local context of $\mathcal{U} \cup \mathcal{S}_\mathcal{U}$ every time, so the k can be modified depending on local information.

Thus, the Pro*k*rustean graph can be used to address the open problem 5 in [22], which calls for a practical representation of variable-order de Bruijn graphs that generates assembly comparable to that of overlap graphs. Consider a specific stratum $S \in \mathcal{S}_\mathcal{U}$. S can be used either as an overlap edge in a overlap graph or a container for a path in a de Bruijn graph. Since S has both information of the suffix-prefix support and frequencies, along

with those information of adjacent maximal repeats, a novel exploration model can be designed. And finally exploration can be represented by a de Bruijn graph of varying order comprising optimized paths. Since the color information is stored only in maximal repeats, the process can be computed within a reasonable space.

5 Discussion

To our knowledge, the Pro*k*rustean graph is the first model to efficiently elucidate how changing k-mer size affects sets of k-mers. The intent is reflected in the name, Pro*k*rustean. In Greek mythology, Procrustes fitted his guests to his bed either by stretching them or cutting their legs. The guests (sequences) of k has experienced a even worse situation; with no such bed, they are unaware of how much they lose their parts (information loss), which is now alleviated by the new bed (the Pro*k*rustean graph) arrived.

Above the Pro*k*rustean graph, a new perspective is suggested: think of sequences and maximal repeats ($\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$) as bins that disjointly contain a set of k-mers of varying k values, co-occurring with each of them. Then, the behaviors of substrings in k-mer-based methods can be interpreted within each bin by adjusting the parameter k and observing the distributions of co-occurring k-mers. We posit that this structure of stratification will provide an entry-point to understand the substring similarity structure of genomic sequences, beneficial both for practitioners and method developers.

The immediate applications of the Pro*k*rustean graph is to compute k-mer-related metrics for a range of k values. We introduced the two applications, counting k-mers and maximal unitigs, where the computation was equivalent to scanning the graph single time. This will allow analyzing k in diverse scenarios, e.g. maximizing the N50 of an assembly with respect to k, or computing the Delta(section 4.2).

A potential next step is the implementation of varying-k methods. Fixing a specific k is by no means a necessary condition, and utilizing two or three k values might seem opportunistic. But the challenge lies in determining how many and which specific values to use because it can lead to infeasible solutions and storage problems. For instance, variable-order de Bruijn graphs encountered this exact issue by managing multiple representations. A competitive method, therefore, should implement a de Bruijn graph with a locally modified order, and the practice should be guided by the correct overlap information stored in the Pro*k*rustean graph. Quantifying the rationale and utilizing them within the local representations of Pro*k*rustean graphs will enable the design of a new methodology, a locally modified assembly graph.

With a broader view, the Pro*k*rustean of a genome can function as a reference database supporting sample querying and genome alignments. Here, the same issue happens with k where matching sequences with samples are often based on k-mers. For example, larger k better specifies the presence or absence of a sample but lowers the matching probability. Multiple k values, based on the sequence representation of locally modified order can suggest a near-optimal approximation of sequences.

It is worthwhile to explore the relationship between a Pro*k*rustean graph and other data structures commonly used in bioinformatics. We have demonstrated that a Pro*k*rustean graph can efficiently be transformed into de Bruijn graphs of all orders and into overlap graphs. Additionally, the Pro*k*rustean graph can easily be used to extract an affix tree, which is the union of two suffix trees built from $\mathcal{U}$ and the reversed strings of $\mathcal{U}$, respectively. To spell out a suffix tree from a Pro*k*rustean graph, one can start by reversing the edge directions and then read the strings of vertices to the right, beginning from the smallest sizes. A suffix tree node is generated upon reaching the end of a vertex, with branching being guided by the possible adjacent vertices. Applying this process in the left direction yields the suffix tree for the reversed sequences. Hence, the vertex cardinality of a Pro*k*rustean graph is similar to that of an affix tree. However, a significant difference lies in the fact that affix links, which connect two vertices of symmetric strings, are not required in a Pro*k*rustean graph.

In terms of practical applications, there is potential for enhancing the construction algorithm of the Pro*k*rustean graph. The primary challenge arises from the fact that the Pro*k*rustean graph is a location-based representation, where integers representing sizes and locations are stored. This leads to both the intermediate and final data being significantly larger than their actual cardinality. Furthermore, on an engineering level, using the ebwt and its derived FM-index, the creation of the wavelet tree and the rank function are the main contributors to time complexity. While we have employed the single-threaded wavelet tree from the SDSL project [11], adopting more advanced methods like those described in [10] could notably enhance performance. Finally, many algorithms that operate on the Pro*k*rustean graph are expected to be highly parallelizable and localizable. Therefore, implementing disk-based computation could enable extremely efficient and high-resolution analysis of the dataset.

Bibliography

[1] M. Abdallah, A. Mahgoub, H. Ahmed, and S. Chaterji. Athena: automated tuning of k-mer based genomic error correction algorithms using language models. *Scientific reports*, 9(1):16157, 2019.

[2] H. N. AlEisa, S. Hamad, and A. Elhadad. K-mer spectrum-based error correction algorithm for next-generation sequencing data. *Computational Intelligence and Neuroscience*, 2022, 2022.

[3] D. Belazzougui and F. Cunial. Fully-functional bidirectional burrows-wheeler indexes and infinite-order de bruijn graphs. In *30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[4] V. Bonnici and V. Manca. Informational laws of genome structures. *Scientific reports*, 6(1):28840, 2016.

[5] J. K. Bonnie, O. Ahmed, and B. Langmead. Dandd: efficient measurement of sequence growth and similarity. *bioRxiv*, pages 2023–02, 2023.

[6] Y. Bussi, R. Kapon, and Z. Reich. Large-scale k-mer-based analysis of the informational properties of genomes, comparative genomics and taxonomy. *PloS one*, 16(10):e0258693, 2021.

[7] R. Chikhi and P. Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014.

[8] A. Cracco and A. I. Tomescu. Extremely fast construction and querying of compacted and colored de bruijn graphs with ggcat. *Genome Research*, pages gr–277615, 2023.

[9] D. D'ıaz-Dom'ınguez, T. Onodera, S. J. Puglisi, and L. Salmela. Genome assembly with variable order de bruijn graphs. *bioRxiv*, pages 2022–09, 2022.

[10] J. Fischer, F. Kurpicz, and M. Löbel. Simple, fast and lightweight parallel wavelet tree construction. In *2018 Proceedings of ALENEX*, pages 9–20. SIAM, 2018.

[11] S. Gog, T. Beller, A. Moffat, and M. Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

[12] D. Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.

[13] R. Islam, R. S. Raju, N. Tasnim, I. H. Shihab, M. A. Bhuiyan, Y. Araf, and T. Islam. Choice of assemblers has a critical impact on de novo assembly of sars-cov-2 genome and characterizing variants. *Briefings in bioinformatics*, 22(5):bbab102, 2021.

[14] K. Kitaya and T. Shibuya. Compression of multiple k-mer sets by iterative spss decomposition. In *21st International Workshop on Algorithms in Bioinformatics (WABI 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

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

[16] 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.

[17] S. C. Manekar and S. R. Sathe. A benchmark study of k-mer counting methods for high-throughput sequencing. *GigaScience*, 7(12):giy125, 2018.

[18] H. Mustafa, M. Karasikov, G. Rätsch, and A. Kahles. Metagraph-mla: Label-guided alignment to variable-order de bruijn graphs. *bioRxiv*, pages 2022–11, 2022.

[19] A. Prjibelski, D. Antipov, D. Meleshko, A. Lapidus, and A. Korobeynikov. Using spades de novo assembler. *Current protocols in bioinformatics*, 70(1):e102, 2020.

[20] T. R. Ranallo-Benavidez, K. S. Jaron, and M. C. Schatz. Genomescope 2.0 and smudgeplot for reference-free profiling of polyploid genomes. *Nat. comm.*, 11(1):1432, 2020.

[21] M. Rautiainen and T. Marschall. Graphaligner: rapid and versatile sequence-to-graph alignment. *Genome biology*, 21(1):253, 2020.

[22] R. Rizzi, S. Beretta, M. Patterson, Y. Pirola, M. Previtali, G. Della Vedova, and P. Bonizzoni. Overlap graphs and de bruijn graphs: data structures for de novo genome assembly in the big data era. *Quantitative Biology*, 7:278–292, 2019.

[23] B. Shariat, N. S. Movahedi, H. Chitsaz, and C. Boucher. Hydavista: towards optimal guided selection of k-mer size for sequence assembly. *BMC genomics*, 15(10):1–8, 2014.

[24] Z. Yang, H. Li, Y. Jia, Y. Zheng, H. Meng, T. Bao, X. Li, and L. Luo. Intrinsic laws of k-mer spectra of genome sequences and evolution mechanism of genomes. *BMC Evolutionary Biology*, 20:1–15, 2020.

Supplemental Material

1 Proofs of theorems

We first prove a simple lemma that is useful to make further discussions concise. The lemma tells if a region is more abundant than its 'parent' string, than it is stratifying or some extension of it is stratifying.

Lemma S1. *For S a string such that $occ_{\mathcal{U}}(S) > 0$, let the string of a region (S, i, j) be more abundant than S. Then (S, i, j) is stratifying in $\mathcal{U}$ or has an extension that is stratifying in $\mathcal{U}$.*

Proof. First, since $|occ_{\mathcal{U}}(str(S, i, j))| > |occ_{\mathcal{U}}(S)|$, by the definition of a stratifying region, (S, i, j) may be stratifying if the additional condition holds: $\forall(S, i', j') \supset (S, i, j)$, $occ_{\mathcal{U}}(str(S, i', j')) = occ_{\mathcal{U}}(S)$. Otherwise, some other extension of (S, i, j) must be stratifying as following. Consider extensions $(S, i-\delta, j+\gamma)$, $0 < \delta+\gamma$. Since the maximum extension is the whole region of S, $occ_{\mathcal{U}}(str(S, i-\delta, j+\gamma)) = occ_{\mathcal{U}}(S)$ eventually holds with some δ and γ. Then there must be some choice of $\delta = \delta^*$ and $\gamma = \gamma^*$ such that increasing each either makes its string occur less or is not valid, i.e. if $i-\delta^* \neq 1$, $|occ_{\mathcal{U}}(S, i-\delta^*, j+\gamma^*)| > |occ_{\mathcal{U}}(S, i-\delta^*-1, j+\gamma^*)| = |occ_{\mathcal{U}}(S)|$ or if $j+\gamma^* \neq |S|$, $|occ_{\mathcal{U}}(S, i-\delta^*, j+\gamma^*)| > |occ_{\mathcal{U}}(S, i-\delta^*, j+\gamma^*+1)| = |occ_{\mathcal{U}}(S)|$. Therefore, $occ_{\mathcal{U}}(S, i-\delta^*, j+\gamma^*)$ is the stratifying extension of (S, i, j). □

The next lemma is the basis for proving subsequent two theorems. Specifically, it is a less intuitive version of the theorem 2.

Lemma S2. *A region (S, i, j) $(S \in \mathcal{U})$ is an occurrence of a maximal repeat if and only if*

1. *(S, i, j) is the whole region of S and $occ_{\mathcal{U}}(S) > 1$ or*
2. *(S, i, j) is stratifying or*
3. *There exists a stratum R such that (S, i, j) is stratifying relatively in R. That is, there exists an extension $(S, i', j') \in \mathrm{occ}_{\mathcal{U}}(R)$ such that $(R, i-i', |R|-(j'-j))$ is stratifying.*

Proof. The forward direction: The first condition is just one obvious scenario of an occurrence of a maximal repeat. Other than the first condition, the string of (S, i, j) is a proper substring of S, so it should be more abundant than its extension. Then Lemma S1 tells that (S, i, j) is stratifying or an extension (S, i_1, j_1) is stratifying. The former case finishes the check by the condition 2, so let us check the latter case that (S, i, j) is not stratifying and (S, i_1, j_1) is stratifying. Then some relative location of (S, i, j) can be defined: let $R_1 := str(S, i_1, j_1)$, which is obviously a stratum by its construction, then $(R_1, i-i_1+1, |R_1|-(j_1-j))$ is the relative position of (S, i, j) in (S, i_1, j_1) that has the same string as (S, i, j). The string of $(R_1, i-i_1+1, |R_1|-(j_1-j))$ must be more abundant than R_1 because the string of (S, i, j) is a maximal repeat and $(R_1, i-i_1+1, |R_1|-(j_1-j))$ cannot be the whole $(R_1, 1, |R_1|)$ considering the construction.

Now we are back to the recursion that Lemma S1 tells that $(R_1, i-i_1+1, |R_1|-(j_1-j))$ is stratifying or an extension is stratifying. If $(R_1, i-i_1+1, |R_1|-(j_1-j))$ is not stratifying the same process above will resulting in finding some R_2 and a relative region of (S, i, j) in it. The process ends at some R_n where the relative region of (S, i, j) being stratifying in it, because $|R_1| > ... > |R_n|$ by the construction and there cannot be an extension of the relative position of (S, i, j) forever. Therefore, picking the R_n and the relative position of (S, i, j) in R_n, the condition 2 satisfies.

The backward direction: If (S, i, j) is stratifying, trivially its string is a stratum by definition. Otherwise, let (S, i, j) be relatively stratifying in an occurrence of a stratum (S, i', j'). Then (S, i, j) is more abundant than S. We show left extension of one unit will make the occurrence decrease, and the right case is symmetric. The left extension $(S, i-1, j)$ is either in (S, i', j') or not, where the latter case happen when $i = i'$.

If $(S, i-1, j) \subset (S, i', j')$, then obviously the relatively stratifying region (S, i, j) in (S, i', j') satisfies $occ_{\mathcal{U}}(S, i-1, j) < occ_{\mathcal{U}}(S, i, j)$. Even if $(S, i-1, j) \not\subset (S, i', j')$, then $i = i'$ that the left extension goes out of the region (S, i', j'). Such case also decreases the occurrence. Let $R := (S, i', j')$ be the stratum relatively including (S, i, j). Then a string of shape cR is less abundant than R for any c. This means for any prefix P of R, cP is less abundant than P. Therefore, the string of (S, i, j), which is also a prefix of R is more abundant than any left extension, and hence the region is an occurrence of a maximal repeat.

Therefore, applying symmetrically to the right case, extending (S, i, j) either left or right makes the occurrence decreased and hence $str(S, i, j)$ is a maximal repeat. □

Theorem 1 (Complete1). *A string R is a maximal repeat if and only if R is a stratum.*

Proof. The forward direction: It is immediate from Lemma S2 that any occurrence (S, i, j) of R satisfies: (1) it is of the whole string that is abundant or (2) it is stratifying or (3) relatively stratifying in some other maximal repeats. (2) is already the condition to be found. For (3) case, the forward proof of Lemma S2 describes there exists some extension (S, i_n, j_n) of a string R_n that includes (S, i, j) as relatively stratifying. Also, R_n is a stratum as described in the proof, so R is a stratum. Lastly for (1) case, its string R must occur at some other place in $\mathcal{U}$ and it must be a proper substring of it, otherwise the string itself is the R. Therefore, the such other occurrence is stratifying or at least relatively stratifying in another stratum and hence R is a stratum, by the logic above.

The backward direction: Any stratum is trivially a maximal repeat, because the definition of stratifying region precisely describes the condition of maximal repeats (and more). □

Theorem 2 (Complete2). *A region (S, i, j) in $S \in \mathcal{U}$ is an occurrence of a maximal repeat*

if and only if

a path from v_S to $v_{S,i,j}$ exists in the Prokrustean graph of $\mathcal{U}$ such that the edge intervals $(i_1, j_1), ..., (i_n, j_n)$ satisfies $i = \Sigma_{t=1}^{n} i_t - (n-1)$.

Proof. The forward direction: Checking the three conditions in Lemma S2 is enough. From the condition 1, if (S, i, j) is the whole string, then voidly the $v_S \in \mathcal{V}_\mathcal{U}$ itself is the path to find. From the condition 2, if (S, i, j) is stratifying, then the edge $e_{S,i,j}$ that is from $v_S \in \mathcal{V}_\mathcal{U}$ to $v_{S,i,j} \in \mathcal{V}_\mathcal{U}$ is the path to find. From the condition 3, if (S, i, j) is not stratifying, then there exists some stratum R_1 in which (S, i, j) is relatively stratifying. It means there exists an edge from $v_{R_1} \in \mathcal{V}_\mathcal{U}$ to $v_{S,i,j} \in \mathcal{V}_\mathcal{U}$. Now, for the $(S, i_1, j_1) \in occ_\mathcal{U}(R_1)$ that made (S, i, j) relatively stratifying in R_1 as in the condition 3, continue the process. Then (S, i_1, j_1) is also an occurrence of a maximal repeat R, it is either stratifying or relatively stratifying in a stratum R_2 by its occurrence (S, i_2, j_2). This process will halt eventually, finding the last stratum R_n whose occurrence (S, i_n, j_n) is stratifying in S and including (S, i, j). Connecting the edges for $S, R_n, R_{n-1}, ..., str(S, i, j)$ forms a path in the Pro*k*rustean graph of $\mathcal{U}$. The formula of i^* is easily verified by the nested regions $(S, i_n, j_n), ..., (S, i_1, j_1), (S, i, j)$. The regions can be converted into edges and their intervals have $i_n, i_{n-1} - i_n + 1, ..., i_1 - i_2 + 1, i - i_1 + 1$ as the left coordinates. Then $\Sigma_{t=1}^{n+1} i^t - n = i_n + (i_{n-1} - i_n + 1) + ... + (i - i_1 + 1) = i$ satisfying the equation. (Note, the number of edges in the path is $n + 1$.)

The backward direction: It is trivial that $str(S, i, j)$ is a stratum, i.e. maximal repeat, and hence (S, i, j) is an occurrence of the maximal repeat in the sequence S. □

Theorem 3 (Compact). *For $\mathcal{U}$ a set of sequences of total length N on the alphabet Σ,*

$$|\mathcal{S}_\mathcal{U}| < N,\ |\mathcal{E}_\mathcal{U}| \leq 2|\Sigma||\mathcal{S}_\mathcal{U}|,\ \textit{and hence } O(|\mathcal{V}_\mathcal{U}| + |\mathcal{E}_\mathcal{U}|) = O(N)$$

Proof. From Theorem 1, we have that strata correspond to maximal repeats, and so the first inequality follows from the well-known fact that the number of maximal repeats (i.e. strata) is less than N ([12]). The second inequality will follow from our showing that for each letter in the alphabet, at most one stratifying region can have the letter at its immediate left among all stratifying regions of a stratum. That is, $\forall c \in \Sigma$, $\forall R \in \mathcal{S}_\mathcal{U}$, $1 \geq |\{(S, i, j) | str(S, i, j) = R \ \&\ e_{S,i,j} \in \mathcal{E}_\mathcal{U} \ \&\ i > 1 \ \&\ S[i-1] = c\}|$. Indeed, if this holds, the case of each letter at the immediate right of a region holds symmetrically, implying that $|\mathcal{E}_\mathcal{U}| \leq 2|\Sigma||\mathcal{S}_\mathcal{U}|$ as desired.

To show this, we argue by contradiction. For a given $c \in \Sigma$, let $e_{S,i,j} \in \mathcal{E}_\mathcal{U}$ has c at its immediate left: $S[i-1] = c$. Assume for contradiction that there exists another stratifying region $e_{S',i',j'} \in \mathcal{E}_\mathcal{U}$ sharing the same string and has the same letter at its left: that is, $str(S, i, j) = str(S', i', j')$ and $S[i-1] = S'[i'-1] = c$. Consider extending two regions together that $(S, i-\delta_l, j+\delta_r)$ and $(S', i'-\delta_l, j'+\delta_r)$ where $\delta_l, \delta_r > 0$. And let δ_l^* and δ_r^* be the largest as possible while preserving string equality: $str(S, i-\delta_l^*, j+\delta_r^*) = str(S', i'-\delta_l^*, j'+\delta_r^*)$. Note that two base regions share c at their immediate left, so $\delta_l^* \geq 1$ is guaranteed and $(S, i, j) \subsetneq (S, i-\delta_l^*, j+\delta_r^*)$. Now, by definition, increasing δ_r^* or δ_l^* will cause the two regions to have different strings or makes at least one region invalid as the end of the string was reached. However, if both extended regions reach both ends of their string, this implies that $(S, i-\delta_l^*, j+\delta_r^*) = (S', i'-\delta_l^*, j'+\delta_r^*) = (S, 1, |S|) = (S', 1, |S'|)$ that S and S' were the same string and hence $(S, i, j) = (S', i', j')$ which is a contradiction. In the other case, at least one region can extend to left or right and get an extended string but the other cannot extend that direction or at least corresponds to a different string. Without loss of generality, say that $(S, i-\delta_l^*, j+\delta_r^*)$ can be extended left. This means that $|occ_\mathcal{U}(str(S, i-\delta_l^*-1, j+\delta_r^*))| < |occ_\mathcal{U}(str(S, i-\delta_l^*, j+\delta_r^*))|$ and hence $|occ_\mathcal{U}(S)| < |occ_\mathcal{U}(str(S, i-\delta_l^*, j+\delta_r^*))|$, and since $(S, i, j) \subsetneq (S, i-\delta_l^*, j+\delta_r^*)$, extending (S, i, j) does not always decrease the occurrence number of corresponding strings, (S, i, j) is not stratifying, which is contradiction.

Theorem 4 (Principle). *For any substring S such that $occ_{\mathcal{U}}(S) > 0$,*

$$S \text{ co-occurs in exactly one string in } \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}.$$

Proof. Assume the theorem does not hold: that S co-occurs in either no string or multiple strings in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$.

Consider the first case, when S co-occurs in no string in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$. Then S always occurs more than any string in which it occurs. So, S must be included in a stratifying region of those strings. Choose one such stratifying region and observe its string, a stratum. S occurs in the stratum, and by assumption its occurrence in the stratum is included in a stratifying region again. This process cannot continue forever because the lengths of the observed strata are strictly decreasing. Therefore, eventually, S is a substring of the last stratum when the process halts, but not occurring in a stratifying region. Clearly, S co-occurs in the last stratum, i.e. occurs in a $|S|$-reflecting region of the stratum, which is contradiction.

Now, consider the second case, when S co-occurs in at least two strings in $\mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$. Consider two occurrences of S in each string, and extend the regions together either to the right or left by one at each step. Since the two strings are different, the strings of the regions should eventually diverge at least once, i.e. the number of occurrences decrease by extension before reaching both ends of both strings. This means S occurs more than at least one of those two superstrings, i.e. does not co-occur in at least one, which is contradiction. □

Corollary 1 (Fixed-k). *For a fixed k,*

$$\bigcup_{\forall S \in \mathcal{U}} \{k\text{-mer in } S\} = \bigsqcup_{\forall S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}} \bigsqcup_{\forall (S,i,j) \in reflect(S,k;\mathcal{U})} \{k\text{-mer in the } k\text{-reflectum } str(S,i,j)\}$$

Proof. By theorem 4, every k-mer co-occurs in exactly one $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$, and such occurrence in S is included in a k-reflecting region of S by definition. Also, every k-reflectum co-occurs in exactly one $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$. Lastly, it is straightforward to check any k-mer co-occurs in exactly one k-reflectum too. Therefore, the right side of the equation captures every k-mer exactly once. □

Corollary 2 (Varying-k). *For a fixed $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$,*

$$\text{The set of substrings that co-occur in } S = \bigsqcup_{\forall k \in 1 \ldots |S|} \bigsqcup_{\forall (S,i,j) \in reflect(S,k;\mathcal{U})} \{k\text{-mer in the } k\text{-reflectum } str(S,i,j)\}$$

Proof. It is obvious that every k-mer K co-occurring in S is captured at the right side of the equation by the previous corollary 1. □

1.1 $reflect(S, k; \mathcal{U})$

The brief algorithm below gets k-reflecting regions of S. We briefly discuss the correctness.

1. Consider stratifying regions of S ordered by their positions.
2. Remove stratifying regions of length less than k.
3. Let the leftmost stratifying region be not the edge case (S, i, j), $1 < i$. Then collect the interval $(1, i + (k - 1) - 1)$.
4. Let some two stratifying regions $(S, i, j), (S, i', j')$ be adjacent that no other stratifying region appears inbetween. And let those intersect less than $k-1$ or do not intersect. Then collect $(j-(k-1)+1, i'+(k-1)-1)$.
5. Let the rightmost stratifying region be not the edge case (S, i, j), $j < |S|$. Then collect the interval $(j - (k - 1) + 1, |S|)$.

Below process of checking correctness is straightforward. First, the collected intervals do not intersect by k or more. Second, every k-mer region not included in a stratifying region is captured by one of the collected intervals. Lastly, since each k-mer captured in the collected intervals occurs the same time as S, and since any possible extension of each interval makes them intersect with a stratifying region at least by k, the interval represents a k-reflecting region of S.

Also, it is obvious that the time complexity is linear to the number of stratifying regions.

2 Applications

We provide descriptions for implementations.

Case	Stratifying regions	w. 3-reflecting region	w. 4-reflecting region	k-mers co-occur in **ACTCCTACG**		
				3-mers	4-mers	count
(1) **No** intersection	A C T C C T A C G	A C T C C T A C G	A C T C C T A C G	ACT, CTC, TCC, CCT, CTA, TAC, ACG	ACTC, CTCC, TCCT, CCTA, CTAC, TACG	**7 →6 (-1)**
(2) **One** intersection	A C T C C T A C G	A C T C C T A C G	A C T C C T A C G	ACT, CTC, TCC	ACTC, CTCC, TCCT	**3→3 (0)**
(3) **Two** intersections	A C T C C T A C G	A C T C C T A C G	A C T C C T A C G	TCC, CCT, CTA	CTCC, TCCT, CCTA, CTAC	**3→4 (+1)**

Fig. S8: Visualization of k-reflecting regions scaling with a progress $k = 3 \rightarrow 4$. Case 1, 2 and 3 cover all possible scenarios of k-reflecting regions in which numbers of substring k-mers change as k increases.

2.1 Count distinct k-mers of range $k = 1 \ldots N$

Below fig. S8 describes each cases of the changes of the number of co-occurring k-mers included in a k-reflecting region by 1 the progress $k \leftarrow k+1$. Each decreases/unchanges/increases by 1 with the progress, when the region intersects with zero/one/or two stratifying regions.

However, the distribution of k-reflecting regions is not static. They can be newly created or merged with another k-reflecting region with the progress $k \leftarrow k+1$. This makes stratifying region-centric observations more tractable as the following theorem.

Theorem S5. *Let $k > 1$. For $C^k(S) :=$ the number of k-mers co-occurring in S,*

$$C^k(S) = C^{k-1}(S) - \mathbf{1}_{No_Intersection(S,k)} + \sum_{\forall(S,i,j)\ stratifying} \mathbf{1}_{Two_Intersections(S,i,j,k)} \tag{5}$$

1. *$No_Intersection(S,k) :=$ A k-reflecting region occupies the whole region of S, i.e. $|$largest stratifying region in S $| < k \leq |S|$*
2. *$Two_Intersections(S,i,j,k) :=$ A k-reflecting region intersects with (S,i,j) on left and another stratifying region on right, i.e. $|(S,i,j) \cap (S,i^r,j^r)|$+1$\leq k \leq |(S,i,j)|$,$|$largest stratifying on right of $(S,i,j)|$*
   *** $|(S,i,j) \cap (S,i^r,j^r)| = max(0, j - i^r + 1)$ where (S,i^r,j^r) is closest stratifying on right of (S,i,j).*

Proof. The conditions are constructive based on the observation of fig. S8.

Note, the ranges of k are dependent on static stratifying regions - largest stratifying region in S, closest and largest stratifying region on right of each stratifying region (S,i,j). So, the computation is not dependent on k. Indeed, the *Algorithm S3* takes $O(|\text{stratifying regions in } S|)$ per each $S \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$. Therefore, counting k-mers for all k's takes $O(|\mathcal{V}_{\mathcal{U}}| + |\mathcal{E}_{\mathcal{U}}|)$ time.

Algorithm S3 Count k-mers of a range

Input: Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, $L \in \mathbb{N}$

Output: Count k-mers for $k = L..N$, $N :=$ maximum sequence length

Set $C, dC, ddC :=$ each list of length N initialized with 0's.
$C[L] =$ Use *Algorithm* 1 for gathering L-reflecta and count L-mers in them.
for $v_S \in \mathcal{V}_\mathcal{U}$ **do**
    Find $e_{max} \in \mathcal{E}_\mathcal{U}$ that represents the largest stratifying region in S
    $ddC[M] -= 1$, where $M := |interval(e_{max})|$
    $ddC[M] += 1$, where $M := |S| + 2$
    Set $e_{rclose} = \varnothing$ # closest region for each e in the iteration below.
    for $e \in \mathcal{E}_\mathcal{U}$ outgoing from v_S and in reverse order **do**
        Set $e_{rmax} = argmax(|interval(e_{rmax})|, |interval(e)|)$ representing the largest stratifying region at right of e.
        $ddC[M] += 1$, where $M := min(|interval(e_{rclose}) \cap interval(e)|, L + 1)$.
        $ddC[M] -= 1$ where $M := min(|interval(e_{rmax})| + 1, |(S, i, j) + 1|)$.
        $e_{rclose} := e$.
    end for
    for $k = L + 1..N$ **do**
        $dC[k] = dC[k - 1] + ddC[k]$
        $C[k] = dC[k - 1] + dC[k]$
    end for
end for

2.2 Count maximal unitigs of de Bruijn graphs of a k range

The strategy is to count the start of each maximal unitig, where three types exist; tips, convergences, and divergences. We focus on divergences and leave the rest in *Algorithm* $S5$. Define $c \in \Sigma$ as an *extending character* of a string S if $occ_\mathcal{U}(Sc) > 0$. Then multiple extending characters of a k-mer correspond to diverging edges in the de Bruijn graph of $k + 1$ built from $\mathcal{U}$. First, we identify such k-mers from the Pro$k$rustean graph of $\mathcal{U}$.

1. A k-mer has multiple extending characters if and only if the k-mer is a suffix of a k-reflectum that is a suffix of a stratum having multiple extending characters.

Above observation is easily verified with corollary 2 and was briefly introduced in fig. 5. Any k-mer K co-occurs in exactly one k-reflectum. The co-occurrence implies the k-reflectum also has multiple extending characters, and since it co-occurs in a string $R\mathcal{U} \cup \mathcal{S}_\mathcal{U}$, it must be occurring at the end of R, or otherwise it is surely included in a stratifying region in R. Lastly, R must be a stratum (not a single occurring sequence) as it has multiple extending characters. The reverse argument holds and hence counting k-mers having multiple extending characters is straightforward: enumerate strata having multiple extending characters, and check if a k-reflecting region is defined at their end. **Note, the extending characters of strata are efficiently counted from its construction *Algorithm* $S5$.

Next, once a k-reflecting region exists at the end of a stratum, the fact does not change through the progress $k \leftarrow k + 1$. So, as in the previous application, we find a strict range for k such that R has a k-reflecting region in its back.

2. For a stratum R, let the length of a stratifying region at the end of R as l, and set $l = 0$ if not applicable. Then for $l < k \leq |R|$, a k-reflecting region exists at the end of R.

Combining observation 1 and 2, we can collect each k range per each stratum where a k-mer having multiple extending characters is implied. So, counting maximal unitigs of de Bruijn graphs of all k's that start after divergences takes $O(|\text{Pro}k\text{rustean}|)$ time. The case of convergences in strata is symmetric to divergences, and the case of tips is the same but consider sequences. Given a desired range of $k := [l : r]$, algorithm S4 counts maximal unitigs for a fixed $k = l$, and is used in algorithm S5 for counting maximal unitigs for $l \leq k \leq r$.

Algorithm S4 Count Maximal Unitigs of a de Bruijn graph of degree k

Input: The Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, $k \in \mathbb{N}$
Output: The number of maximal unitigs of the de Bruijn graph of $\mathcal{U}$ of degree k

1: Set $cnt := 0$
2: **for** $v_S \in \mathcal{V}_\mathcal{U}$ such that $S \in \mathcal{U}$ **do**
3: **If** $reflect(v_S, k)$ shows leftmost k-reflecting region starts at 1 **then** $cnt \leftarrow cnt + 1$
4: **end for**
5: **for** $v_S \in \mathcal{V}_\mathcal{U}$ such that $S \in \mathcal{S}_\mathcal{U}$ **do**
6: **if** $size(v_S) > k - 1$ **then**
7: Run $reflect(v_S, k)$ to get $l-$interval/$r-$interval:= leftmost/rightmost k-reflecting regions intervals.
8: **If** $l-$interval starts with 1 **AND** 0 left extending character of S **then** $cnt \leftarrow cnt + 1$
9: **If** $l-$interval starts with 1 **AND** multiple left extending characters of S **then** $cnt \leftarrow cnt + 1$
10: **If** $r-$interval ends with $|S|$ **AND** multiple(m) right extending characters of S **then** $cnt \leftarrow cnt + m$
11: **else if** $size(v_S) = k - 1$ **then**
12: **If** multiple left extending characters of S **then** $cnt \leftarrow cnt + 1$
13: **Else If** multiple(m) right extending characters of S **then** $cnt \leftarrow cnt + m$
14: **end if**
15: **end for**
16: Return cnt

Algorithm S5 Count maximal unitigs of de Bruijn graphs of k-mers of a k range

Input: Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, $L \in \mathbb{N}$
Output: Count maximal unitigs for $k = L..N$, $N :=$ maximum sequence length

1: Set $C, dC :=$ each list of length N initialized with 0's.
2: $C[L] =$ run *Algorithm S4* to count maximal unitigs at L.
3: **for** $v_S \in V$ such that $S \in \mathcal{U}$ **do**
4: **If** $|interval(e :=$ outgoing edge of interval (1,*))$| \geq L - 1$ **then** $dC[|interval(e)| + 2]$+=1
5: **end for**
6: **for** $v_S \in V$ such that $S \in \mathcal{S}_\mathcal{U}$ **do**
7: **if** 0 left extending character of S **then**
8: **If** $|interval(e :=$ outgoing edge of interval (1,*))$| \geq L - 1$ **then** $dC[|interval(e)| + 2]$+=1
9: -1 to $dC[size(v_S) + 2]$
10: **else if** multiple (m) left extending character of S **then**
11: **If** $|interval(e :=$ outgoing edge of interval (1,*))$| \geq L - 1$ **then** $dC[|interval(e)| + 2]$+=1
12: **If** 1 right extending character of S **then** $dC[|size(v_S)| + 2]$-=1 **else** $dC[|size(v_S)| + 1]$-=1
13: **end if**
14: **if** multiple (m) right extending character of S **then**
15: **If** $|interval(e :=$ outgoing edge of interval (*,$|S|$))$| \geq L - 1$ **then** $dC[|interval(e)| + 2]$+=m
16: $dC[size(v_S) + 2]$-=m
17: **end if**
18: **for** $k = L + 1..N$ **do**
19: $C[k] = C[k - 1] + dC[k]$
20: **end for**
21: **end for**
22: Return C

The algorithm S4 implements the observation 1. for tips/convergences/divergences in the previous page. Note, the line3 and line7 are actually modified in the software code so that only leftmost and rightmost stratifying regions are considered. In that way, algorithm S4 runs in $O(\mathcal{V}_\mathcal{U})$ time.

The algorithm S5 implements the observations 2. for tips/convergences/divergences. The list dC represents $dC[x] = C[x] - C[x - 1]$.

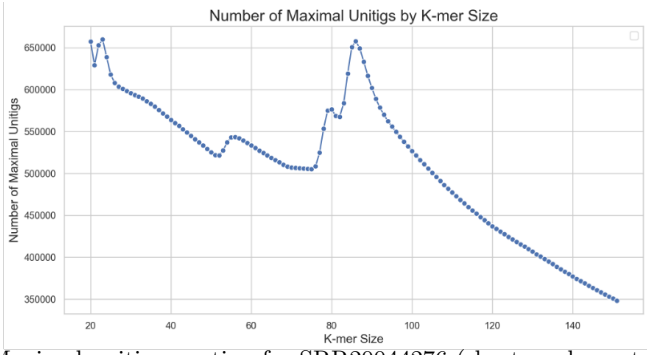

Fig. S9: Maximal unitig counting for SRR20044276 (short reads, metagenome).

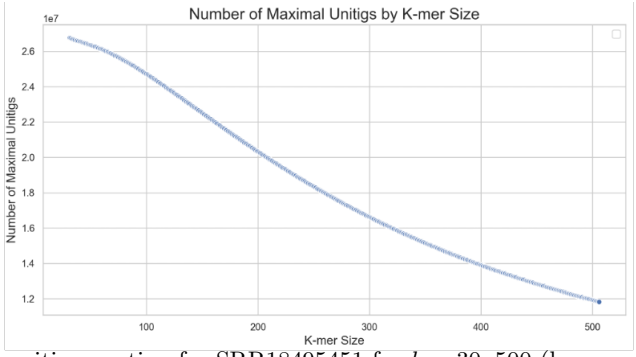

Fig. S10: Maximal unitig counting for SRR18495451 for $k = 30..500$ (long reads, metagenome).

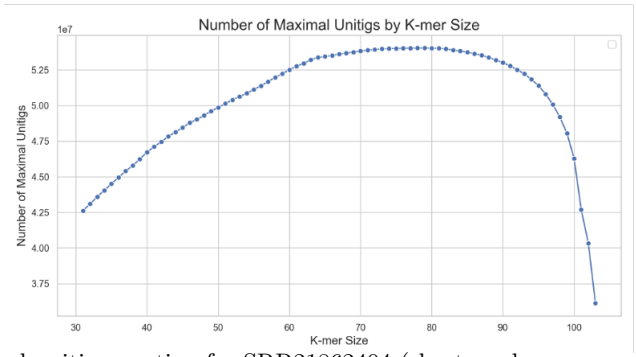

Fig. S11: Maximal unitig counting for SRR21862404 (short reads, rna-seq, homo sapiens).

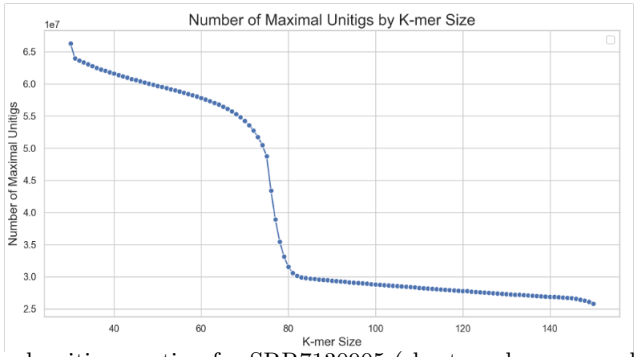

Fig. S12: Maximal unitig counting for SRR7130905 (short reads, rna-seq, homo sapiens).

2.3 Extract overlap graphs and de Bruijn graphs

Recall that below algorithm extracts an overlap graph from a Pro*k*rustean graph.

Algorithm S6 Extract the overlap graph of $\mathcal{U}$

Input: Pro*k*rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, $L \in \mathbb{N}$
Output: The overlap graph of $\mathcal{U}$ of the threshold L (least overlap length)

1: Make $(V, E) = (\{v_S \in \mathcal{V}_\mathcal{U} | S \in \mathcal{U}\}, \varnothing)$.
2: **for** $v_S \in \mathcal{V}_\mathcal{U}$ such that $S \in \mathcal{S}_\mathcal{U}$ and $size(v_S) \geq L$ **do**
3: Set $V_l = \{v_{S_l} \in V | S$ is a prefix of $S_l\}$, $V_r = \{v_{S_r} \in V | S$ is a suffix of $S_r\}$.
4: $E = E \cup V_l \times V_r$ where the new edges are labeled $size(v_S)$.
5: **end for**
6: Return (V, E).

Recall that any k-mer co-occurs in exactly one $S \in \mathcal{U} \cup \mathcal{S}_\mathcal{U}$, and hence shares the right extension if the k-mer is the suffix of S. Therefore, extracting a de bruijn graph of order k is equivalent to identifying all k-mers co-occurring in strings in $\mathcal{U} \cup \mathcal{S}_\mathcal{U}$.

A transformation from the Pro*k*rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$ to the de Bruijn graph $(V^k_\mathcal{U}, E^k_\mathcal{U})$ is defined in a way that it implies the task that can be done in $O(|\text{Pro}k\text{rustean}|)$ time. First, we define a k-mer K as a *left k-extension* of a string S if a suffix of K and a prefix of S match by length $k-1$. A *right k-extension* of S is symmetrically defined. Second, we define a prefix stratum P of a stratum R as a *front k-descendant* if P is the smallest among prefixes that are strata of length at least $k-1$. A *back k-descendant* is symmetrically defined. The name *descendant* is because a path in the Pro*k*rustean graph R to P is observed and sizes of those strata decrease. Then the transformation is as following:

1. $V^k_\mathcal{U}$: k-mer regions in k-reflecting regions of $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$.
2. $E^k_\mathcal{U}$: Let a k-mer region (S, i, j) in a k-reflecting region.
   (a) Let (S, i, j) intersects with a stratifying region by $k-1$ on its right and the string of the intersection is not a stratum. Then $((S, i, j), (S', i', j')) \in E^k_\mathcal{U}$ is the only outgoing edge of (S, i, j) such that $str(S, i+1, j+1) = str(S', i', j')$ and (S', i', j') is in a k-reflecting region.
   Symmetrically, an incoming edge of (S, i, j) can be found with a stratifying region on left.
   (b) Let (S, i, j) intersects with a stratifying region by $k-1$ on its right and the string of the intersection is a stratum R. Then (S, i, j) is registered as a 'left context' of R. It forms multiple edges with all 'right contexts' of R.
   Symmetrically, the 'right contexts' are obtained.
   (c) For each co-occurring $(k+1)$-mer region in $\mathcal{V}_\mathcal{U}$ such that the $k-1$ string in the middle is not a stratum, an edge is made from prefix to suffix.
   ** Note1: The cases (a)-(c) cover all possible extensions of a k-mer co-occur in a string S.
   ** Note2: At (b), checking whether the intersection is a stratum R or not is decided by checking *front* or *back* descendents of the stratum defined by the intersecting stratifying region. The descendents can be constructed by scanning the Pro*k*rustean graph once.

3 Pro*k*rustean graph construction

Data Preparation Each dataset was downloaded from NCBI and merged into a single text file where each sequence ends with a terminal symbol '$'. And then using grlBWT, one of ebwt computing codes, we gathered the ebwt file convertible to an FM-index. Then the Pro*k*rustean graph was built with the codes in prokrustean. However, instead of introducing the ebwt version, which is a bit complicated, the core concepts and required properties are described with two suffix trees. The ebwt version is a mere optimized version.

3.1 Two suffix trees approach

A Pro*k*rustean graph can be built with two generalized suffix trees - one built with $\mathcal{U}$ and the other with reverse strings of $\mathcal{U}$ - or an affix tree of $\mathcal{U}$. We briefly introduce how stratification is connected to the suffix tree structure, which clearly reveals the straightforward way of obtaining a Pro*k*rustean graph. However, the straightforward method has less applicability because building a suffix tree requires about $20N$ bytes for the cumulative sequence length N of $\mathcal{U}$. The consequential algorithms of Pro*k*rustean graph construction use the ebwt input that implicitly navigate the suffix tree structure of $\mathcal{U}$.

Let $\mathcal{U}^r$ be the set of reversed strings of $\mathcal{U}$. Let $T_{\mathcal{U}}$ and $T_{\mathcal{U}^r}$ be each the generalized suffix tree built with $\mathcal{U}$ and $\mathcal{U}^r$, respectively. Also, let $node(S) \in T_{\mathcal{U}}$ the node representing S and $node^r(S^r) \in T^r_{\mathcal{U}}$ the node representing S^r. Below fact is well-known in the context of an affix tree.

Fact. If both $node(S)$ and $node^r(S^r)$ exist, then R is a maximal repeat.

Now, let $(node(S), node(S'))$ be an edge in $T_{\mathcal{U}}$. Let an edge $e_{S,i,j}$ be in a Pro*k*rustean graph of $\mathcal{U}$. Then three stratification scenarios can be described by $T_{\mathcal{U}}$ and $T^r_{\mathcal{U}}$.

1. If $i = 1$, then $node(str(S, i, j))$ in $T_{\mathcal{U}}$ has an edge representing $S[j + 1..|S|]$.
2. If $j = |S|$, then $node^r(str^r(S, i, j))$ in $T^r_{\mathcal{U}}$ has an edge representing the reverse of $S[1..i - 1]$.
3. If $i \neq 1$ & $j \neq |S|$, then both scenarios above hold.

Below intuitive observations describes information-wise inclusion between the suffix trees and a Pro*k*rustean graph. That is, every stratifying relationship is depicted on edge(s) of the suffix trees. For example, the scenario 1 can be briefly proved by the definition of stratification. Since (S, i, j) can be extended on right and hence its string is a maximal repeat(stratum), there exists a $node(str(S, i, j))$ having an edge branching with $S[j + 1]$. Now, the terminal node is $node(S)$ because otherwise, say $node(S')$ is the terminal node, S' is a substring of S. It contradicts to the fact that (S, i, j) is stratifying because there must be an extension (S, i', j') of (S, i, j) whose string is S', meaning (S, i, j) is relatively stratifying in (S, i', j').

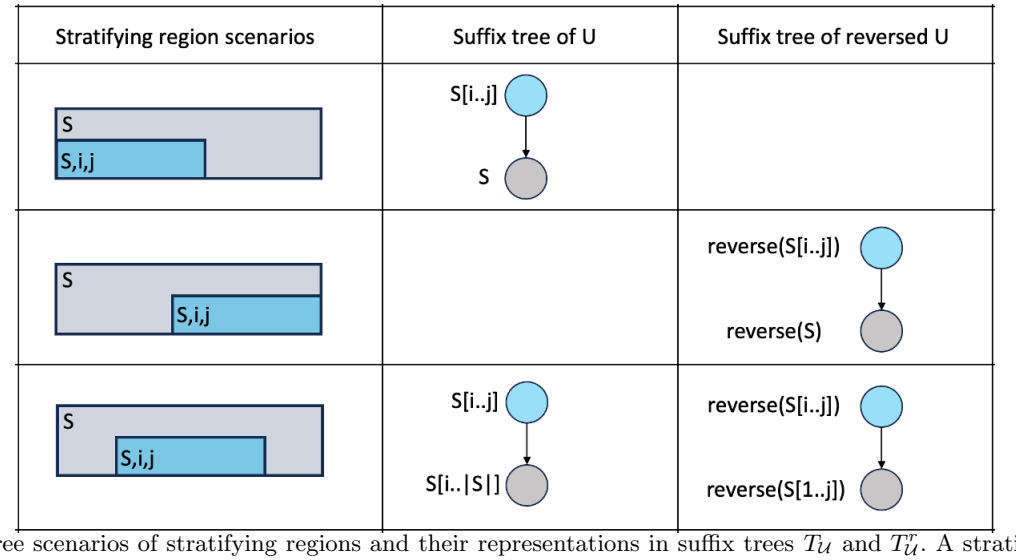

Fig. S13: Three scenarios of stratifying regions and their representations in suffix trees $T_{\mathcal{U}}$ and $T^r_{\mathcal{U}}$. A stratifying region (S, i, j) can be placed at each end of S or in the middle of S. Since the existence of a node in $T_{\mathcal{U}}$ implies right-maximality and the existence of a node in $T^r_{\mathcal{U}}$ implies left-maximality, starting from the string of (S, i, j), the left over substring of $S[j + 1..|S|]$ or $S[1..i - 1]$ is represented by some edge in $T_{\mathcal{U}}$ or $T^r_{\mathcal{U}}$.

However, the reverse of the observations do not hold, meaning an additional condition is required to decide which edges of the suffix trees imply stratifying relationships. For example, even if the first scenario in fig. S13

holds that $(node(S), node(str(S,i,j)))$ is an edge in $T_{\mathcal{U}}$, an edge of $e_{S,1,*}$ may not exist in the Pro*k*rustean graph. The further development is introduced in theorem S6.

3.2 Deciding which edges in a suffix tree imply stratifying relationships.

Definition S8. *For a string S occurring in $\mathcal{U}$, $\overline{S} \in \mathcal{U} \cup \mathcal{S}_{\mathcal{U}}$ satisfies S co-occurs in $\overline{S}$.*

Theorem 4 says $\overline{S}$ is unique for any string S.

Definition S9. *Let $\sigma_l, \sigma_r \in \Sigma$ and $R \in \mathcal{S}_{\mathcal{U}}$ such that $occ_{\mathcal{U}}(\sigma_l R), occ_{\mathcal{U}}(R\sigma_r) > 0$. Attaching the letters to R, eg. $\sigma_l R$ we mean a new concatenated string that σ_l is on the immediate left of R.*

- *σ_l and σ_r are <u>exclusive</u> over R if and only if $|occ_{\mathcal{U}}(\sigma_l R)| = |occ_{\mathcal{U}}(\sigma_l R\sigma_r)| = |occ_{\mathcal{U}}(R\sigma_r)|$.*
- *σ_r is <u>right-exclusive</u> to σ_l if and only if $|occ_{\mathcal{U}}(\sigma_l R)| = |occ_{\mathcal{U}}(\sigma_l R\sigma_r)|$.*
- *σ_l is <u>left-exclusive</u> to σ_r if and only if $|occ_{\mathcal{U}}(\sigma_l R\sigma_r)| = |occ_{\mathcal{U}}(R\sigma_r)|$.*

Theorem S6. *Let R be a maximal repeat in $\mathcal{U}$ and $\sigma_l, \sigma_r \in \Sigma$ such that $|occ_{\mathcal{U}}(\sigma_l R)|, |occ_{\mathcal{U}}(R\sigma_r)| > 0$. Also, let $node(S)$ denotes the node representing S, when the tree of the node is clear.*

1. *If no $\sigma \in \Sigma$ is <u>left-exclusive</u> to σ_r over R, $node(R_r)$, the child node of $node(R)$ of $T_{\mathcal{U}}$ found from the branch σ_r, represents $\overline{R\sigma_r}$. Therefore, $(R_r, 1, |R|)$ is stratifying.*
2. *If no $\sigma \in \Sigma$ is right-exclusive to σ_l over R, $node(R_l^r)$, the child node of $node(R_l^r)$ of $T_{\mathcal{U}^r}$ found from the branch σ_l, represents the reverse string of $\overline{\sigma_l R^r}$. Therefore, $((R_l^r)^r, |R|, |(R_l^r)^r| - |R|)$ is stratifying.*
3. *If σ_l and σ_r are exclusive over R, consider $node(R_r)$, the child node of $node(R)$ of $T_{\mathcal{U}}$ found from the branch σ_r, and $node(R_l^r)$, the child node of $node(R^r)$ of $T_{\mathcal{U}^r}$ found from the branch σ_l, meaning R_r and the reverse of R_l^r matches the prefix and suffix by R. Merge R_r and the reverse of R_l^r by overlapping the R and generate S. Then $(S, |R_l^r| - |R|, |R_l^r|)$ is stratifying.*

Proof. 1. $node(R_r)$ found from the branch σ_r of $node(R)$ satisfies $R_r = \overline{R\sigma_r}$. First, R_r is a maximal repeat because it is represented by a node (right maximal), and no $\sigma \in \Sigma$ is left-exclusive to σ_r over R, meaning extending on the left of R, which is a prefix of R_r makes the occurrence decreased. Second, $(R_r, 1, |R|)$ is stratifying, because $R\sigma_r$ co-occurs in R_r. For contradiction, if a larger region is an occurrence of a stratum, then $R\sigma_r$ should co-occur in the stratum instead of R_r.

2. This is symmetric to 1., only carefully considering the reverse strings.

3. This is also similar to 1., but the exception is that R_r and $(R_l^r)^r$ are not maximal repeats. If R_r and $(R_l^r)^r$ are merged by overlapping the R, meaning concatenating $(R_l^r)^r[1..|R|]$, R, $R + R_r[|R|+1..|R_r|]$ and obtain S, both $R\sigma_r$ and $\sigma_l R$ co-occur in S. It is because σ_l and σ_r are exclusive, so extending R to S will not decrease the occurrence. Therefore, $(S, |R_l^r| - |R|, |R_l^r|)$ is stratifying. □