Prokrustean Graph: A substring index supporting rapid enumeration across a range of k-mer sizes

Adam Park
Department of Computer Science and Engineering, Pennsylvania State University, PA, USA

David Koslicki
Department of Computer Science and Engineering, Department of Biology, and the Huck Institutes of the Life Sciences, Pennsylvania State University, PA, USA

Abstract

Despite the widespread adoption of k-mer-based methods in bioinformatics, a fundamental question persists: How can we quantify the influence of k sizes in applications? With no universal answer available, choosing an optimal k size or employing multiple k sizes remains application-specific, arbitrary, and computationally expensive. The assessment of the primary parameter k is typically empirical, based on the end products of applications which pass complex processes of genome analysis, comparison, assembly, alignment, and error correction. The elusiveness of the problem stems from a limited understanding of the transitions of k-mers with respect to k sizes. Indeed, there is considerable room for improving both practice and theory by exploring k-mer-specific quantities across multiple k sizes.

This paper introduces an algorithmic framework built upon a novel substring representation: the Pro$k$rustean graph. The primary functionality of this framework is to extract various k-mer-based quantities across a range of k sizes, but its computational complexity depends only on maximal repeats, not on the k range. For example, counting maximal unitigs of de Bruijn graphs for $k = 10, \ldots, 100$ takes just a few seconds with a Pro$k$rustean graph built on a read set of gigabases in size. This efficiency sets the graph apart from other substring indices, such as the FM-index, which are normally optimized for string pattern searching rather than for depicting the substring structure across varying lengths. However, the Pro$k$rustean graph is expected to close this gap, as it can be built using the extended Burrows-Wheeler Transform (eBWT) in a space-efficient manner.

The framework is particularly useful in pangenome and metagenome analyses, where the demand for precise multi-k approaches is increasing due to the complex and diverse nature of the information being managed. We introduce four applications implemented with the framework that extract key quantities actively utilized in modern pangenomics and metagenomics.

Code implementing our data structure and algorithms (along with correctness tests) is available at `https://github.com/KoslickiLab/prokrustean`.

2012 ACM Subject Classification Applied computing → Computational biology

Keywords and phrases k-mer, k-mer spectra, genome assembly, pangenomics, metagenomics, FM-index, BWT

Digital Object Identifier 10.4230/LIPIcs.WABI.2024.YY

Supplementary Material `https://github.com/KoslickiLab/prokrustean`

Acknowledgements This research was supported by the NIH grant R01GM146462

© Adam Park and David Koslicki;
licensed under Creative Commons License CC-BY 4.0
Workshop on Algorithms in Bioinformatics 2024.
Editors: John Q. Open and Joan R. Access; Article No. YY; pp. YY:1–YY:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

As sequencing read data and the number of reference genomes increase each year, k-mer-based methods continue to gain popularity in bioinformatics. Although fixing the substring length, k, facilitates computational efficiency by splitting sequences into manageable pieces, the influence of k size is still poorly understood, impeding further developments. There is inherent information loss for any choice of k because k-mers represent only a subset of all possible substrings of their original sequences. Quantifying this information loss is crucial for virtually every application in evaluating how k sizes affect their outcomes. However, lacking any theoretical or practical framework, the quantification is mostly based on application outputs, where noisy factors and computational burdens obscure the analysis.

Two common ad-hoc practices prevail today. First, researchers arbitrarily choose a "folklore" k value based on previous empirical results. Second, in multi-k approaches, they iterate through k-mers of many k sizes to achieve accuracy comparable to brute-force methods, which sacrifices the computational benefits of utilizing k-mers.

In genome analysis, distinct k-mers serves as a proxy for the structure of a genome, but the increasingly large sizes of genome data today limit experiments to only a few selected k values [6, 30, 10]. A recent study about pangenome complexity suggests that the number of distinct k-mers across all k sizes provides additional insights [7].

Metagenome comparative analyses, such as those using Bray-Curtis dissimilarity based on frequencies of k-mers from reads, often face resource constraints due to the large number of comparisons, while the demand for utilizing multiple k sizes increases [16, 28, 27]. More conventionally, methods that rely on k-mer spectra struggle to identify the "best" k size for their experiments [2, 36, 21].

Genome assemblers are probably the most sensitive to the choice of k size and are facing computational challenges employing multi-k approaches with k-mer de Bruijn graphs. The choice of k size is crucial in de novo assemblers, yet it still relies on heuristic methods [20, 12, 29]. Additionally, the creation of multi-k de Bruijn graphs remains a challenging and mostly theoretical endeavor [31, 33, 15]. While metagenome assemblers require multi-k strategies more due to their complex inputs, no significant breakthroughs have occurred in the k selection since the introduction of metaSPAdes, which uses arbitrary k sizes [26].

Meanwhile, modern substring indexes do not serve as an ideal framework for this problem. Substring indexes, e.g. suffix trees/arrays, BWT, FM-indexes, and others, are built on similar substring representations, characterized by the LCP array. Their support for string pattern queries, preserving lossless information of substrings, can be interpreted as storing k-mers of any sizes intact, making their applications to k-mer-related tasks appealing [4, 9, 1]. However, the main unsolvable issue is that the computational cost strictly increases with the queried k sizes, as is obvious in "order changes" in multi-k de Bruijn graphs [4].

Motivated by the pressing need to develop a framework for quantifying the influence of k size, we introduce the Pro$k$rustean graph, a simple hierarchical structure that recursively captures maximal repeats of given sequences. Then an algorithmic approach enables rapid enumeration of k-mer contexts above the graph, shifting the focus of enumeration from the k sizes to maximal repeats. This significantly reduces the information space and computational complexity, distinguishing the Pro$k$rustean graph from other substring indexes.

The following task characterizes the essence of the Pro$k$rustean graph: Given a set of sequences, populate a vector for $k = 1 \ldots N$ with the number of distinct k-mers. This task takes time and space linear to the number of maximal repeats, i.e. the complexity is independent of the queried range of k sizes $[1, N]$. See Figure 1 which summarizes our theme.

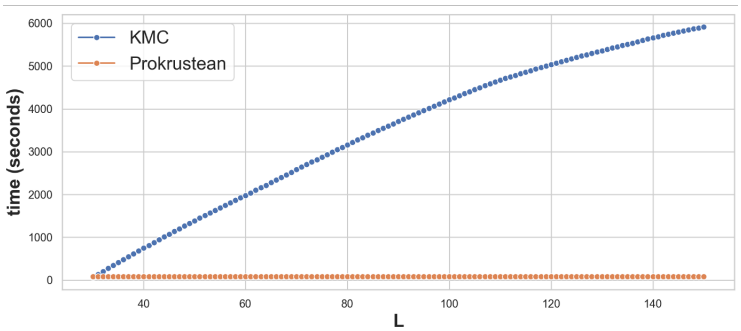


Figure 1 Counting distinct k-mers for $k = 30, \ldots, L$ of a dataset (4.5gb, ERR3450203) with Pro$k$rustean graphs and KMC, the state-of-the-art k-mer counting tool [22]. While the Pro$k$rustean graph counts k-mers of all k sizes in a single run, KMC should iterate through k sizes.

▶ **Problem 1.** *How can we efficiently compute k-mer quantities for multiple k sizes? Let k-mer quantities those defined with k-mers frequently used in genomics:*

- ***Counts**, e.g. the number of distinct k-mers, DandD [7].*
- ***Frequencies**, e.g. the Bray-Curtis dissimilarity between two sequence sets [28].*
- ***Extensions**, e.g. the number of maximal unitigs in a k-mer de Bruijn graph [23].*
- ***Occurrences**, e.g. vertex degrees of the overlap graph identified by length-k overlaps[35].*

Let Σ be the alphabet, $\mathcal{R}$ the set of maximal repeats in the sequences, and k_{min} be a threshold such that $\mathcal{R}(k_{min})$ is the set of maximal repeats of length at least k_{min}.

▶ **Objective 2.** *Given a set of sequences, build a framework that computes k-mer quantities for all k sizes in time and space linear to $O(|\Sigma| \cdot |\mathcal{R}|)$.*

The framework is progressively constructed by improving computational complexity:

- Section 2 introduces a substring index, the Pro$k$rustean graph, and its underlying concepts.
- Section 3 introduces the computation of *k-mer quantities* of a single k size in $O(|\Sigma| \cdot |\mathcal{R}|)$.
- Section 4 extends the computation to all k sizes, maintaining the complexity of $O(|\Sigma| \cdot |\mathcal{R}|)$.
- Section 5 focuses on the practical range of k sizes of at least k_{min}, improving the complexity to $O(|\Sigma| \cdot |\mathcal{R}(k_{min})|)$, and presents results of various implementations.

2 Pro$k$rustean graph: A compact hierarchy of maximal repeats

2.1 Basic Notations

Let $S \in \Sigma^*$ represent a finite-length string, and $\mathcal{U} \subset \Sigma^*$ for a set of strings. Consider the following preliminary definitions:

▶ **Definition 3.**

- *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: $str(S, i, j) := S_i S_{i+1} \ldots S_j$.*
- *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:*

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

- *S is a maximal repeat in $\mathcal{U}$ if it occurs at least twice and more than any of its extensions: $|occ_{\mathcal{U}}(S)| > 1$, and for all S' a superstring of S, $|occ_{\mathcal{U}}(S)| > |occ_{\mathcal{U}}(S')|$. 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. Consider the description of repeats in three sequences: ACCCT, GACCC, and TCCCG. Maximal repeats, ACCC and CCC, are useful to cover all repeating patterns, but their total occurrences can be overwhelmingly numerous. Consider the following two statements:

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 "occurrence information," while the second case uses 4 units, yet preserves the same meaning. Theoretically, the latter reduces the quadratic number of occurrences of maximal repeats to linear. Figure 2 depicts the rule.

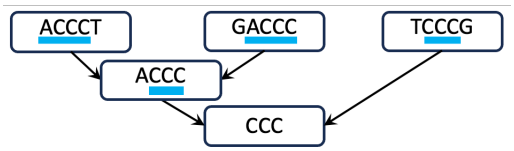


Figure 2 Stratification. Given three strings ACCCT, GACCC, and TCCCG, maximally matching subregions (indicated with blue underlining) are used to label terminal nodes of each edge. This process repeats hierarchically, hence the name *stratification*.

▶ **Definition 4.** *A region (S, i, j) of a substring S of a sequence in $\mathcal{U}$ is stratifying in $\mathcal{U}$ if:*

$$|occ_{\mathcal{U}}(str(S, i, j))| > |occ_{\mathcal{U}}(S)| \tag{1}$$

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

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

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. For example, $(\text{ACCCT}, 1, 4)$ (i.e. ACCC) is stratifying in $\mathcal{U} = \{\text{ACCCT, GACCC, TCCCG}\}$. Also, the captured string ACCC has the region $(\text{ACCC}, 2, 4)$ (i.e. CCC) stratifying in $\mathcal{U}$. However, the region $(\text{ACCCT}, 2, 4)$ (i.e. CCC) is not stratifying in $\mathcal{U}$ because its extension $(\text{ACCCT}, 1, 4)$ (i.e. ACCC) appears 2 times in $\mathcal{U}$ while the parent ACCCT appears 1 time in $\mathcal{U}$.

2.3 Pro$k$rustean Graph

The Pro$k$rustean graph of $\mathcal{U}$ is simply a graph representation of stratification processed on $\mathcal{U}$, i.e. vertexes represent strings and edges represent stratifying regions. First, to construct the vertex set, all maximal repeats identified through stratification are collected.

▶ **Definition 5.** *Define $\mathcal{R}_{\mathcal{U}}$ the string set captured through stratification on $\mathcal{U}$ as follows:*

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

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

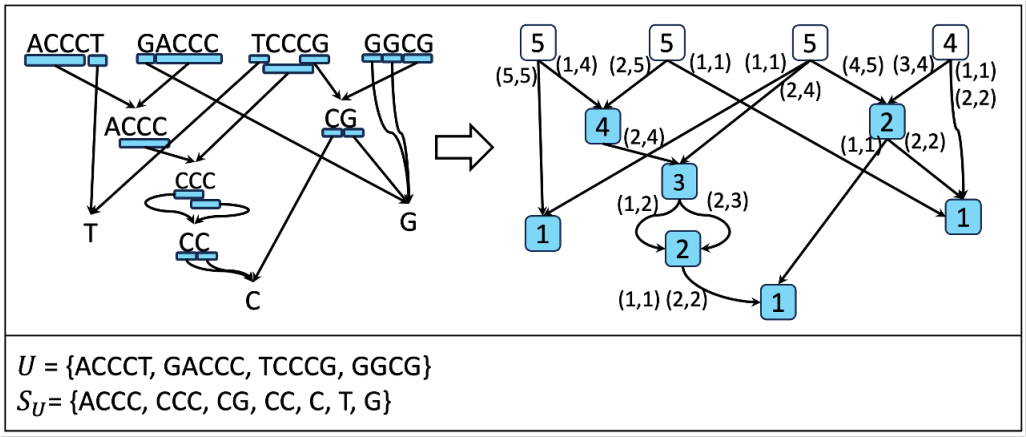


Figure 3 The Pro*k*rustean graph of $\mathcal{U}$={ACCCT,GACCC,TCCCG,GGCG}. Stratification is depicted in the left graph. Then the Pro*k*rustean graph on the right uses integer labels to store the region information. The number in each vertex is the size of the string, and the interval of each edge is a stratifying region in its source vertex.

▶ **Definition 6.** *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* $S \in \mathcal{U} \cup \mathcal{R}_\mathcal{U}$
- $\mathcal{E}_\mathcal{U}$: $e_{S,i,j} \in \mathcal{E}_\mathcal{U}$ *if and only if* $(S,i,j) \in \bigcup_{S \in \mathcal{U} \cup \mathcal{R}_\mathcal{U}} \{(S,i,j) | (S,i,j) \text{ is stratifying}\}$
- $e_{S,i,j} \in \mathcal{E}_\mathcal{U}$ *directs from* v_S *to* $v_{str(S,i,j)}$
- *for* $v_S \in \mathcal{V}_\mathcal{U}$, $size(v_S) = |S|$
- *for* $e_{S,i,j} \in \mathcal{E}_\mathcal{U}$, $interval(e_{S,i,j}) = (i,j)$

Check Figure 3 to see how stratification is labeled in a Pro*k*rustean graph. Next, two propositions ensure that all maximal repeats are packed into the optimally compact model.

▶ **Proposition 7** (Complete). *$\mathcal{R}_\mathcal{U}$ is the set of maximal repeats of $\mathcal{U}$, i.e. a string R is captured from the construction of $\mathcal{R}_\mathcal{U}$ if and only if R is a maximal repeat.*

Proof. Obviously, all strings in $\mathcal{R}_\mathcal{U}$ are maximal repeats by the definition of stratifying regions. The reverse argument that $\mathcal{R}_\mathcal{U}$ captures all maximal repeats is given in Appendix C. ◀

Therefore, the vertex set is the union of sequences and maximal repeats. Trivially assuming that $|\mathcal{U}| \ll |\mathcal{R}_\mathcal{U}|$, i.e., there are plenty of maximal repeats, we have $O(|\mathcal{V}_\mathcal{U}|) = O(|\mathcal{R}_\mathcal{U}|)$.

▶ **Proposition 8** (Compact). *The number of incoming edges per vertex is at most $2|\Sigma|$, so*

$$|\mathcal{E}_\mathcal{U}| \leq 2|\Sigma||\mathcal{V}_\mathcal{U}|.$$

Proof. This rule is analogous to the extension representation of suffix trees. Each incoming edge of a vertex v_S corresponds to a unique extension of S. Thus, two distinct stratifying regions capturing the same maximal repeat and sharing the same extension yield a contradiction; i.e., these duplicated extensions define a maximal repeat in which a single stratifying region merges those two into one. Appendix C covers the details. ◀

The inequality implies that the size of a Pro*k*rustean graph is $O(|\Sigma||\mathcal{R}_\mathcal{U}|) =: O(|\text{Pro}k\text{rustean}|)$, emphasizing that its size depends on the number of maximal repeats, assuming $|\Sigma|$ is constant, as is typical in genomic sequences. Denoting N as the accumulated length of sequences in $\mathcal{U}$, it is a well-known fact that $|\mathcal{R}_\mathcal{U}| < N$ [19].

3 Computing k-*mer quantities* for a single k

This section introduces the computation of k-mer quantities for a fixed k size, which requires $O(|\text{Pro}k\text{rustean}|)$ time and space. This process is termed *complementation*, serving as a mechanism that accesses k-mers by locally counteracting stratification.

3.1 Complementation

Complementation guides how to efficiently access k-mers from the hierarchy of stratification. Consider the example given in Figure 2, recalling that blue regions covered locally maximal similarities. For a given k-mer size, a *complementary* region is a maximal region that covers k-mers that are not included in a blue region. See the red regions underlining in Figure 4 where two different k-mer sizes are used to show complementary regions.

The key idea is that reading k-mers in red regions is equivalent to enumerating distinct k-mers. For example, the $k = 3$ case in Figure 4, ACC is included in a red region in ACCCC and does not appear again in any other red region, while occurring in two blue regions in nodes ACCCT and GACCC. Note that a red region may include multiple k-mers (e.g. TCCCG at $k = 4$), so k-mers can be efficiently enumerated through *complementation*.

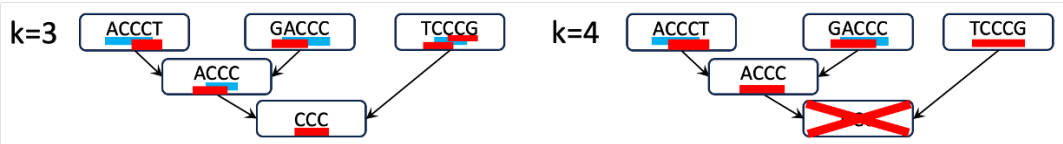


Figure 4 *Complementation.* Given sizes of $k = 3$ and 4, find "complementary regions" in which k-mers are not in covered regions. Blue underlining depicts covered regions, and red underlining depicts regions that should be complemented. Every k-mer appears exactly once within a red/complementary region, while k-mers can appear arbitrarily many times in blue/covered regions.

The figure also reveals a basic example of how structural transitions influence the set of k-mers when the k-mer size increases from $k = 3$ to $k = 4$. While all three sequences are connected through the two vertices in the left graph ($k = 3$), the right graph ($k = 4$) shows an isolation of TCCCG from others.

Now, we define the complementary regions based on a core property: co-occurrence.

▶ **Definition 9.** *A region (S, i, j) of a string S is* <u>*k-co-occurring in $\mathcal{U}$*</u> *if:*
for each k-mer K in $str(S, i, j)$,

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

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

$$|occ_{\mathcal{U}}(K)| > |occ_{\mathcal{U}}(S)|. \tag{4}$$

In words, the string of a k-co-occurring region maximally contains k-mers that co-occur with S. Co-occurrence is extremely valuable information in applications. For example, co-occurring k-mers form a unitig (simple path) in the de Bruijn graph of order k.

Complementation is designed to enable the enumeration of k-mers and their contexts with efficiency—linear relative to the number of stratifying regions of S. Define a computation <u>*complement*(S, k)</u> that identifies k-co-occurring regions as follows: Scan through the stratifying regions in S and capture regions of maximal length (at least k) that do not overlap with any stratifying region by more than $k - 1$. Refer to Figure 5 for concrete examples.

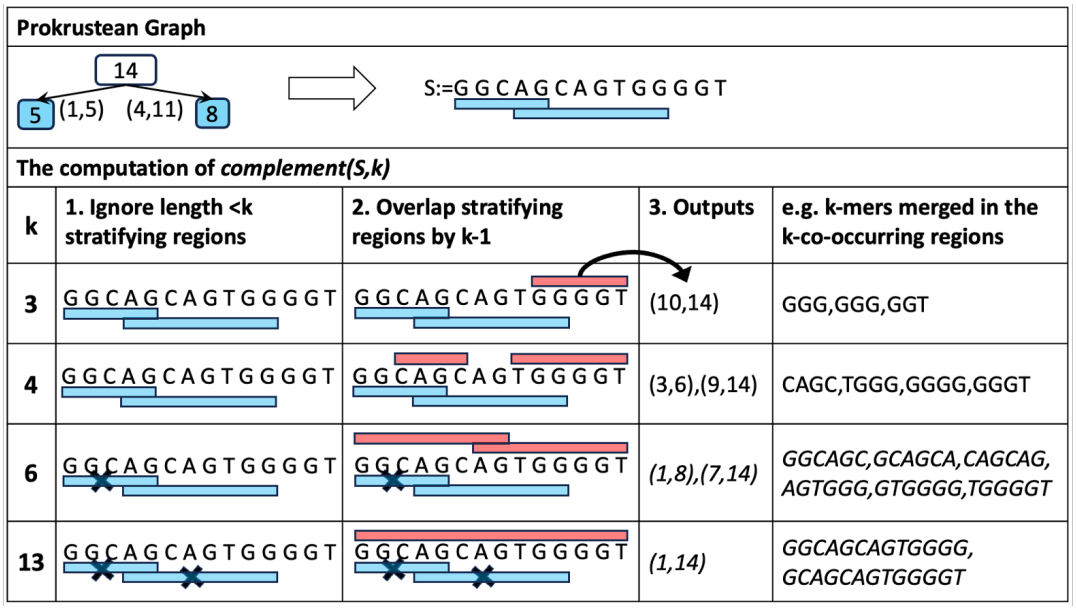


Figure 5 *complement*(S, k) - computing k-co-occurring regions (red) in S of varying k sizes. The rule is to cover every k-mer that is not covered by two stratifying regions (blue). Note that a k-co-occurring region can appear on the overlap of stratifying regions. For example, the region (S, 3,6) at CAGC in the second row ($k = 4$) is 4-co-occurring because it overlaps both blue regions by 3.

▶ **Proposition 10.** *Regions identified by complement*(S, k) *are the set of k-co-occurring regions in S, and the computation takes $O(|$stratifying regions in $S|)$ time.*

Proof. By definition, a k-co-occurring region contains k-mers that co-occur with S, but its extension does not. This is equivalent to a region being maximal, satisfying it does not overlap any stratifying region by k or more. In such a region, any k-mer does not occur more than in S; otherwise, it would be included in a stratifying region. Since the region is maximal, its extensions overlap a stratifying region by at least k. The computation requires $O(|$stratifying regions in $S|)$ time as it simply scans through stratifying regions. ◀

3.2 Application: Counting distinct k-mers of a single k size

We leverage the Pro$k$rustean graph and k-co-occurring regions to count distinct k-mers. Below algorithm counts k-mers merged in k-co-occurring regions.

Algorithm 1 Count distinct k-mers of a single k

Input: A Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, a k size
Output: The number of distinct k-mers in $\mathcal{U}$

```
1 for v_S ∈ V_U do
2     for (i, j) in complement(v_S, k) do
3         cnt += j − i + 2 − k
4 return cnt
```

The complexity is $O(|$Pro$k$rustean$|)$ because the nested loop in line 1 and 2 is in total $O(\mathcal{E}_\mathcal{U})$, as the total number of k-co-occurring regions is linear to the total stratifying regions (Proposition 10). The correctness is verified by Proposition 21 in the next section.

4 Framework: Computing k-*mer quantities* for all k sizes

This section introduces the computation of k-mer quantities for all k sizes, preserving the $O(|\text{Pro}k\text{rustean}|)$ time and space as seen in the single k size Section 3.

4.1 Theory of co-occurrence

We first introduce the main theoretical discovery of this work, which can be fully leveraged by the Pro$k$rustean graph, thus distinguishing it from other substring indexes. Roughly speaking, the Pro$k$rustean graph represents a "flipped and merged" version of the affix tree—the union of the suffix tree and the reversed suffix tree. This shift in perspective offers new insights into co-occurrence grouping: All substrings of varying lengths can be disjointly grouped by their co-occurrences, and co-occurring substrings together exhibit the same behavior in applications.

▶ **Definition 11.** *A string S co-occurs in 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 12** (Principle)**.** *For any substring S such that $occ_{\mathcal{U}}(S) > 0$,*

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

Proof. Assume the theorem does not hold that S co-occurs in either no string or multiple strings in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. Consider the first case, when S co-occurs in no string in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. S anyway occurs in some string, so S always occurs more than any string in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. So, S must be included in a stratifying region of those strings. Choose one such stratifying region and observe its string, a maximal repeat. By assumption it occurs more than the maximal repeat too, and hence is included in a stratifying region. Continuing this process cannot last forever because the lengths of the observed maximal repeats are strictly decreasing. Therefore, eventually, S is a substring of the last maximal repeat when the process halts, not occurring in a stratifying region. Clearly, S co-occurs in the last maximal repeat, which is contradiction.

Now, consider the second case, when S co-occurs in at least two strings in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. Consider two occurrences of S in two strings in which S co-occur, and extend the regions together either to the right or left by one at each step. Since the two regions are in different strings, the strings of the regions should eventually diverge at least once, i.e. the number of occurrences decrease by an 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. ◀

This theorem indicates that all substrings in $\mathcal{U}$, regardless of their length, are disjointly assigned to sequences and maximal repeats ($\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$) by co-occurrence. Additionally, the correctness of Algorithm 1 is derived from this principle.

▶ **Proposition 13.** *Any k-mer in $\mathcal{U}$ occurs in exactly one k-co-occurring region in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$.*

Proof. Theorem 12 states that for any k-mer, there exists exactly one $S \in \mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ in which the k-mer co-occurs. The k-mer must be found in a k-co-occurring region in S, and in no other; otherwise, the k-mer cannot co-occur in S which is a contradiction. ◀

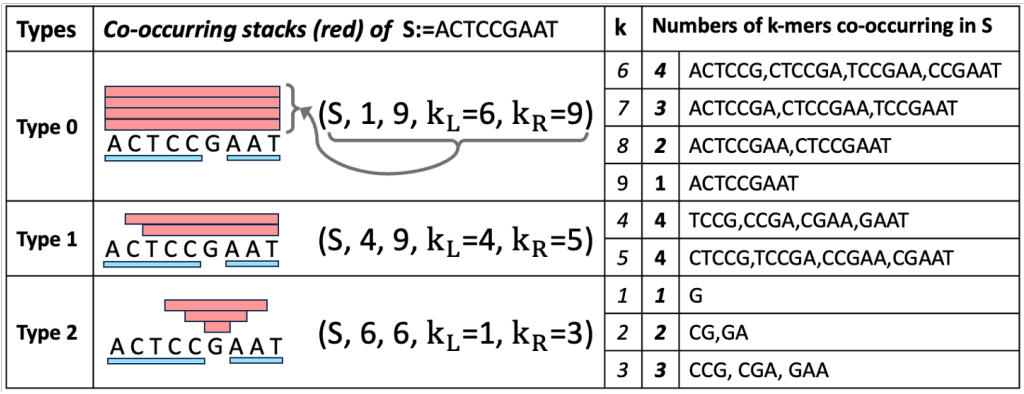


Figure 6 Three basic examples of *co-occurring stacks* of S given two stratifying regions. The stacks (red) of three types are represented by 5-tuple expressions. The Type 0 case has length-9 regions for $k = 6, 7, 8, 9$ with the change rate -1: The number of co-occurring k-mers are $4, 3, 2, 1$.

4.2 Grouping co-occurring k-mers for a k range

To achieve k-range-independent complexity, k-mers should be carefully grouped.

▶ **Definition 14.** *A co-occurring stack of S is a compact representation of a group of k-co-occurring regions of a k range:*

$$(S, i, j, k_L, k_R)$$

which expresses that each (S, i_k, j_k) is a k-co-occurring region of S for $k = k_L, \ldots, k_R$

$$\text{where } i_k = \begin{cases} 1 & \text{if } i = 1 \\ i + k_L - k & \text{otherwise,} \end{cases}, \; j_k = \begin{cases} |S| & \text{if } j = |S| \\ j - k_L + k & \text{otherwise.} \end{cases}$$

Figure 6 shows simple patterns of *co-occurring stacks*. We can leverage the patterns to enjoy computational benefits if all substrings of all lengths can be covered with a feasible number of *co-occurring stacks*, as implied by Proposition 21 below.

Its proof reveals that *maximal co-occurring stacks* can be classified into **Type 0/1/2**: Each stack is homogeneous, with every region (S, i_k, j_k) overlapping the same **0/1/2** stratifying region(s) by $k-1$. Within each stack, the change rate of k-mers is consistent, i.e. the number of k-mers co-occurring in each region in the stacks changes by -1, 0, and +1 for Type 0, 1, and 2, respectively, when k increases by +1.

▶ **Proposition 15.** *Define $\mathcal{S}_\mathcal{U}$:= a collection of co-occurring stacks that express all k-co-occurring regions found in $\mathcal{U} \cup \mathcal{R}_\mathcal{U}$ each exactly once. Then,*

$$|\mathcal{S}_\mathcal{U}| \leq |\mathcal{V}_\mathcal{U}| + 2|\mathcal{E}_\mathcal{U}|.$$

Proof. (1) A k-co-occurring region and a k'-co-occurring region in S can be collected in the same *co-occurring stack* if they overlap the same stratifying region(s) by $k-1$. (2) We aim to maximally capture such homogeneous regions in a *stack*, hence *stacks* are classified by Types 0-2. (3) Maximal Type 0 stacks each contains whole regions, hence occur $|\mathcal{V}_\mathcal{U}|$ times. (4) A *maximal* Type 1 or 2 stack always contains the largest region overlapping a stratifying region of length k_R by $k_R - 1$. (5) Thus, for each side of each stratifying region of length l, there can be at most one *co-occurring stack* containing a l-co-occurring region as the largest region, so maximal Type 1 or 2 cases occur at most $2|\mathcal{E}_\mathcal{U}|$ times. Therefore, $|\mathcal{S}_\mathcal{U}|$ is upper bounded by $|\mathcal{V}_\mathcal{U}| + 2|\mathcal{E}_\mathcal{U}|$. (1)-(5) are further extended in Appendix C. ◀

**Note, enumerating $\mathcal{S}_\mathcal{U}$ takes $O(|\text{Pro}k\text{rustean}|)$ time and space. (Appendix D.1).

4.3 Application: Counting distinct k-mers of all k sizes

The following algorithm counts distinct k-mers of all k sizes using *co-occurring stacks*. The first and second loops each require $O(|\mathcal{S}_\mathcal{U}|)$ and $O(k_{max})$ time, respectively. We can safely assume $k_{max} < |\mathcal{S}_\mathcal{U}|$ for genomic sequences, even if k_{max} is the maximum length in $\mathcal{U}$, because $\mathcal{U}$ typically includes many sequences in practice. Therefore, the algorithm running for all k sizes requires $O(\text{Pro}k\text{rustean})$ time and space.

Algorithm 2 Count k-mers of a range of k sizes

Input: A Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, k_{max}
Output: The number of distinct k-mers in $\mathcal{U}$ for $k = 1, \ldots, k_{max}$

1 Make three 0 vectors of length k_{max}: C, ∂C, and $\partial\partial C$.
2 Compute co-occurring stacks $\mathcal{S}_\mathcal{U}$.
3 **for** $(v_S, i, j, k_L, k_R) \in \mathcal{S}_\mathcal{U}$ **do**
4 $\quad C[k_L]{+}{=}j - i - k + 2$
5 $\quad \delta = \mathbf{1}_{\{1<i\}} + \mathbf{1}_{\{j<|size(v_S)|\}} - 1$
6 $\quad \partial\partial C[k_L + 1]{+}{=}\delta$
7 $\quad \partial\partial C[k_R]{-}{=}\delta$
8 **for** $k = 2 \ldots k_{max}$ **do**
9 $\quad \partial C[k]{+}{=}\partial C[k-1]{+}\partial\partial C[k]$
10 $\quad C[k]{+}{=}C[k-1]{+}\partial C[k]$
11 **return** C

4.4 Application: Computing Bray-Curtis dissimilarities of all k sizes

The Bray-Curtis dissimilarity, in the context of k-mers, is a frequency-based metric that measures the dissimilarity between sets. For two sequence sets, $\mathcal{U}_A$ and $\mathcal{U}_B$, from samples A and B, we first construct the Pro$k$rustean graph of $\mathcal{U}_A \bigsqcup \mathcal{U}_B$, labeling the origins of sequences. Then the computation for $k = 1, \ldots, k_{max}$ requires $O(|\text{Pro}k\text{rustean}|)$ time and space. This approach scales to multiple samples, resulting in $O(|\text{samples}| \cdot |\text{Pro}k\text{rustean}|)$ complexity.

$$BC(\mathcal{U}_A, \mathcal{U}_B, k) = \sum_{\forall kmer \text{ in } \mathcal{U}_A \cup \mathcal{U}_B} \frac{min(freq_A(kmer), freq_B(kmer))}{freq_A(kmer) + freq_B(kmer)}$$

Algorithm 3 Compute Bray-Curtis dissimilarities of a range of k sizes

Input: A Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$ where $\mathcal{U} = \mathcal{U}_A \bigsqcup \mathcal{U}_B$, k_{max}
Output: Bray-Curtis dissimilarities between samples A and B for $k = 1, \ldots, k_{max}$

1 Make 0 vectors of length k_{max}: D,∂D,$\partial\partial D$, N,∂N,$\partial\partial N$. (Denominator, Numerator)
2 Compute frequencies $freq_A(v_S)$ and $freq_B(v_S)$ for all $v_S \in \mathcal{V}_\mathcal{U}$.
3 Compute co-occurring stacks $\mathcal{S}_\mathcal{U}$.
4 **for** $(v_S, i, j, k_L, k_R) \in \mathcal{S}_\mathcal{U}$ **do**
5 $\quad$ # Imitate the value assignments to C, $\partial\partial C$ at lines 4-7 in Algorithm 2.
6 $\quad$ Assign to N, $\partial\partial N$ with all values weighted with $min(freq_A(v_S), freq_B(v_S))$
7 $\quad$ Assign to D, $\partial\partial D$ with all values weighted with $freq_A(v_S) + freq_B(v_S)$
8 **for** $k = 2 \ldots k_{max}$ **do**
9 $\quad$ # Imitate the value assignments to C using ∂C,$\partial\partial C$ at lines 9-10 in Algorithm 2.
10 $\quad$ Assign to N using ∂N, $\partial\partial N$
11 $\quad$ Assign to D using ∂D, $\partial\partial D$
12 **return** N/D

4.5 Application: Counting maximal unitigs of all k sizes

A maximal unitig of a de Bruijn graph, or a vertex of a compacted de Bruijn graph [14], is a simple path that cannot be extended further, reflecting the topology of the graph. The counting idea is that maximal unitigs are implied by tips, convergences, and divergences, which are indicated by Type 0 or 1 co-occurring stacks. For example, assume that S in Figure 6 has multiple right extensions. Every k-co-occurring region in the Type 0 and 1 co-occurring stacks, covering $k = 4, \ldots, 9$, includes a suffix k-mer of S that co-occurs in S. It implies that each suffix k-mer of S is "valid" for all $k = 4, \ldots, 9$, hence S contributes to a divergence in de Bruijn graphs of orders $k = 4, \ldots, 9$.

Algorithm 4 Count maximal unitigs of de Bruijn graphs a range of k sizes

Input: A Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, k_{max}
Output: Numbers of maximal unitigs of de Bruijn graphs of orders $k = 1, \ldots, k_{max}$

```
Make 0 vectors of length k_max: C,∂C.
Compute numbers of left/right extensions left(v_S) and right(v_S) for all v_S ∈ V_U.
Compute co-occurring stacks S_U.
for (v_S, i, j, k_L, k_R) ∈ S_U do
    if i = 1 and (left(v_S) = 0 or left(v_S)>1) # tips or convergence then
        C[k_L]+=1
        ∂C[k_R]−=1
    if j = size(v_S) and right(v_S) > 1 # divergence then
        C[k_L]+=right(v_S)
        ∂C[k_R]−=right(v_S)
for k = 2...k_max do
    C[k]+=C[k − 1]+∂C[k]
return C
```

4.6 Application: Computing vertex degrees of overlap graph

Overlap graphs are widely used in genome and metagenome assembly along with de Bruijn graphs. A critical downside of overlap graphs is their quadratic growth with the number of reads ($O(|\mathcal{U}|^2)$). However, the Pro$k$rustean graph includes the overlap graph as a subgraph, in a different hierarchical form. This structure allows the computation of vertex degrees of the overlap graph in $O(|\text{Pro}k\text{rustean}|)$ time and space.

The overlap graph of $\mathcal{U}$ has the vertex set $\mathcal{U}$, and an edge $(v_S, v_{S'})$ if and only if a suffix of S matches a prefix of S'. Define $pref(v_R \in \mathcal{V}_\mathcal{U})$ as the number of R occurring as a prefix in strings in $\mathcal{U}$, which is easily computed through the recursive structure.

$$pref(v_R \in \mathcal{V}_\mathcal{U}) := \sum_{e=(v_{R'}, v_R) \in \mathcal{E}_\mathcal{U}, interval(e)=(1,*)} pref(v_{R'})$$

Starting from $v_S \in \mathcal{V}_\mathcal{U}$, explore a suffix path, meaning recursively choose edges that are suffix stratifying regions. Then, sum up $pref(v_*)$ for all vertices encountered along the path. Since $pref(v_R)$ represents the occurrence of R as a prefix, the summed quantity equals the outgoing degree of vertex S in the overlap graph of $\mathcal{U}$. Symmetrically, incoming degrees can be computed by defining $suff(v_R)$. The recursion can be strategically organized so that vertices are visited only once when computing all overlap degrees of $\mathcal{U}$, thereby reducing the overall complexity to $O(|\text{Pro}k\text{rustean}|)$.

5 Implementations and Results

The Pro*k*rustean graph construction and four applications were implemented and tested with various datasets. Datasets were randomly selected to represent a broad range of sizes, sequencing technologies, and biological origins. To demonstrate the scalability of the Pro*k*rustean graph, 16 human pangenome and 3,500 metagenome references were used. Diverse sequencing datasets were collected too, including two metagenome short-read datasets, one metagenome long-read dataset, and two human transcriptome short-read datasets. The full list of datasets is provided in Appendix A.

5.1 Complexity improvement with k_{min}

The complexity $O(|\text{Pro}k\text{rustean}|) = O(|\Sigma| \cdot |\mathcal{R}_{\mathcal{U}}|)$ can be improved by dropping maximal repeats of length below k_{min}. The threshold limits the length of stratifying regions to $\geq k_{min}$, capturing a subgraph of the Pro*k*rustean graph, and hence achieves $O(|\text{Pro}k\text{rustean}|) = O(|\Sigma| \cdot |\mathcal{R}_{\mathcal{U}}(k_{min})|)$ time while computing *k-mer quantities* for $k = k_{min}, \ldots, k_{max}$. Figure 7 shows how the graph size of short reads of length 151 decreases as k_{min} increases.

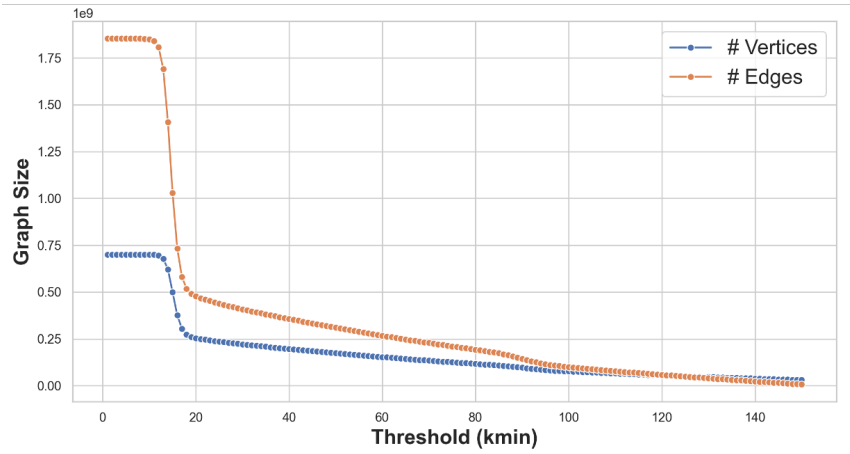


Figure 7 Sizes of Pro*k*rustean graphs (number of vertices and edges) versus k_{min} values for a short read dataset ERR3450203. The size of the graph drops around $k_{min} = [11, ..., 19]$, meaning the maximal repeats and stratifying regions are dense when k_{min} is under 11. Since the graph eventually becomes fully disconnected, the number of edges falls below that of the vertices ($100 \leq k_{min}$).

5.2 Pro*k*rustean graph construction with k_{min}

The Pro*k*rustean graph can be built using the extended Borrow-Willer transform in $O(N)$ time and space where N denotes the cumulative sequence length of $\mathcal{U}$ (Appendix B). The required time and memory are highly dependent on the repeats that should be captured.

dataset	reads	N	k_{min}	time	mem	output
SRR20044276(metagenomic)	0.94M	72M	20	13s	200MB	40MB
ERR3450203(metagenomic)	55M	4.5B	30	28min	11GB	2.7GB
SRR18495451(metagenomic/long)	0.83M	11.3B	30	90min	43GB	14GB
SRR7130905 (human/rna-seq)	45M	6.8B	30	39min	18GB	4GB
SRR21862404 (human/rna-seq)	336M	22.3B	30	99min	37GB	5.9GB

Table 1 Performances of Pro*k*rustean graph construction from eBWTs. Symbols M, B, MB, and GB mean millions, billions, megabytes, and gigabytes. We chose $k_{min} = 30$ for most cases which is common in practices of genome assembly. N (input base pairs) and outputs are comparable.

It is observed that $|\text{Pro}k\text{rustean}| \ll N$, with $|\mathcal{V}_\mathcal{U}| < 0.2N$ and $|\mathcal{E}_\mathcal{U}| < 0.5N$ with short reads. Additionally, for 3500 metagenome references with $k_{min} = 10$, $|\mathcal{V}_\mathcal{U}| < 0.01N$ and $|\mathcal{E}_\mathcal{U}| < 0.03N$ were observed. Figure 8 shows how references of pangenome and metagnome grow. Again, the growth of the graph follows the growth of maximal repeats in the sequence set $(O(|\mathcal{R}_\mathcal{U}|))$.

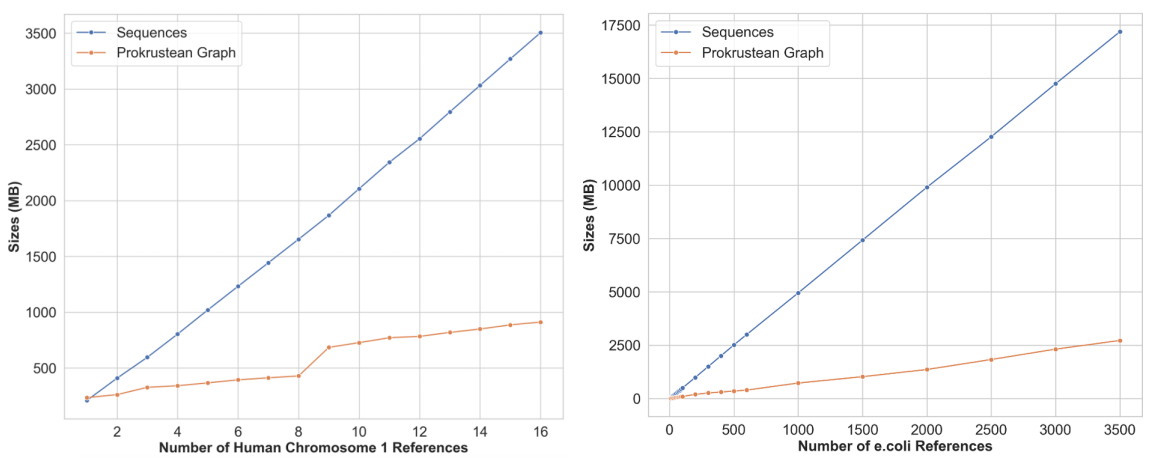


Figure 8 Sizes of Pro$k$rustean graphs as a function of sequences added. Left: Pro$k$rustean graphs constructed using an accumulation of 16 human chromosome 1 references. Right: same but generated using 3500 E. coli references exibiting an increased growth rate reflecting E. coli species diversity.

Note that in the subsequent sections covering the results of four applications, no library performs the exact same tasks, making performance comparisons inherently "unfair." Readers are encouraged to focus on the overall time scale to gauge the efficiency of the Pro$k$rustean graph. Additionally, discrepancies in the outputs of the comparisons may arise due to practice-specific configurations in libraries, such as the use of only ACGT and canonical k-mers in KMC. Consequently, we tested correctness on GitHub using brute-force implementations.

5.3 Result: Counting distinct k-mers for $k = k_{min}, \ldots, k_{max}$

Distinct k-mers are commonly used to measure the complexity and structures of pangenomes and metagenomes [34, 7, 8]. We employed KMC [22], an optimized k-mer counting library designed for a fixed k size , and used it iteratively for all k values. This "unfair" comparison emphasizes the limitations of current practices in counting k-mers across various k sizes.

Dataset	k	KMC			Prokrustean Graph		
		time	memory	threads	time (compute)	memory	threads
SRR20044276	1...150	5m	837mb	22	**0.36s** (0.3s)	80mb	8
ERR3450203	1...150	70m	12gb	22	**106s** (15s)	11gb	8
SRR18495451	30...50000	days	-	22	**88s** (13s)	24gb	8
SRR7130905	30...150	66m	12gb	22	**46s** (6s)	8.5gb	8
SRR21862404	30...150	112m	13gb	22	**74s** (8s)	13.8gb	8

Table 2 Counting distinct k-mers with Pro$k$rustean graphs (Algorithm 2) compared with KMC. KMC had to be executed iteratively causing the computational time to increase steadily as the range of k increases. The running time of KMC for each k took about 1-3 minutes.

5.4 Result: Computing Bray-Curtis dissimilarities for $k_{min}, \ldots, k_{max}$

In metagenomics, Bray-Curtis dissimilarities measure sample differences using k-mers in reads. Selecting appropriate k sizes for varying biological contexts remains contentious [16, 28, 27], but comparing k-mer frequencies is computationally intensive even with purpose-built libraries [13, 5]. Below, in Figure 9, computing dissimilarities between four metagenome samples shows that no k size is "correct"; instead, new insights are obtained from multiple k sizes. This task took about 10 minutes with the Pro$k$rustean graph of four samples of 12 gigabase pairs in total.

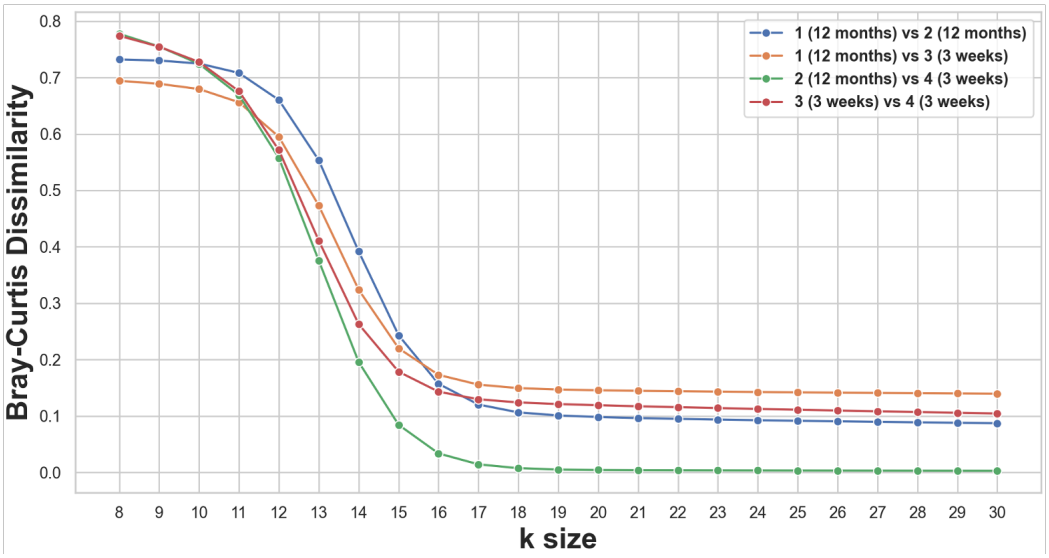


Figure 9 Bray-Curtis dissimilarities between four samples derived from infant fecal microbiota, which were studied in [28]. Samples include two from 12-month-olds and two from 3-week-olds. The ranks between values shift as k changes, e.g. the dissimilarity between sample 2 and 4 is highest at $k = 8$ but lowest at $k = 21$. Note that diverse k sizes (10, 12, 21, 31) are actively utilized in practice.

5.5 Result: Counting maximal unitigs for $k_{min}, \ldots, k_{max}$

Maximal unitigs, simple paths in de Bruijn graphs, effectively represent the topology of these graphs and are particularly meaningful in genome assemblers. Understanding their changes across various k sizes has huge potential for multi-k methods and analyses. However, no library supports this computation directly, so we utilized GGCAT [14], a mature library optimized for a fixed k size in de Bruijn graph construction. The table below displays the comparison, and Figure 10 illustrates some outputs.

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

Table 3 Counting maximal unitigs with GGCAT and Prokrustean Graph(Algorithm 4). The efficiency shown in the compute column is emphasized, where only a few seconds are required for large datasets. The GGCAT was executed for k sizes iteratively. The running time for each k took about 1-5 minutes for GGCAT. Both methods used 8 threads.

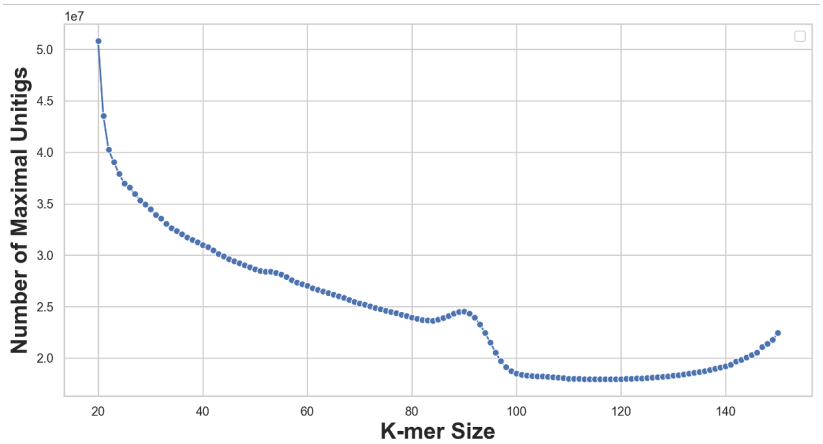


Figure 10 The number of maximal unitigs in de Bruijn graphs of order $k = 20, \ldots, 150$ of metagenomic short reads (ERR3450203). A complex scenario is revealed: The decrease of the numbers fluctuates in $k = 30, \ldots, 60$, and the peak around $k = 90$ corresponds to the sudden decrease in maximal repeats. Lastly, further disconnections increase contigs and eventually make the graph completely disconnected.

5.6 Result: Counting vertex degrees of overlap graph of threshold k_{min}

Overlap graphs are primarily used in genome assembly along with de Bruijn graphs, and their overlap information is particularly crucial for metagenomics in read classification [11, 32, 24, 3] and contig binning [35, 25]. Vertex degrees are often interpreted as (abundant-weighted) read coverage in metagenomic samples, which exhibit high variances due to species diversity and abundance variability. A common computational challenge is the quadratic growth of overlap graphs relative to the number of reads $O(\mathcal{U}^2)$. However, limiting the task to vertex degree counting, $O(|\text{Pro}k\text{rustean}|)$ time and space are required (Section 4.6). With the Pro$k$rustean graph of 27 million short reads (4.5 gigabase pairs in total), the computation generating Figure 11 used 8.5 gigabytes memory and 1 minute to count all vertex degrees.

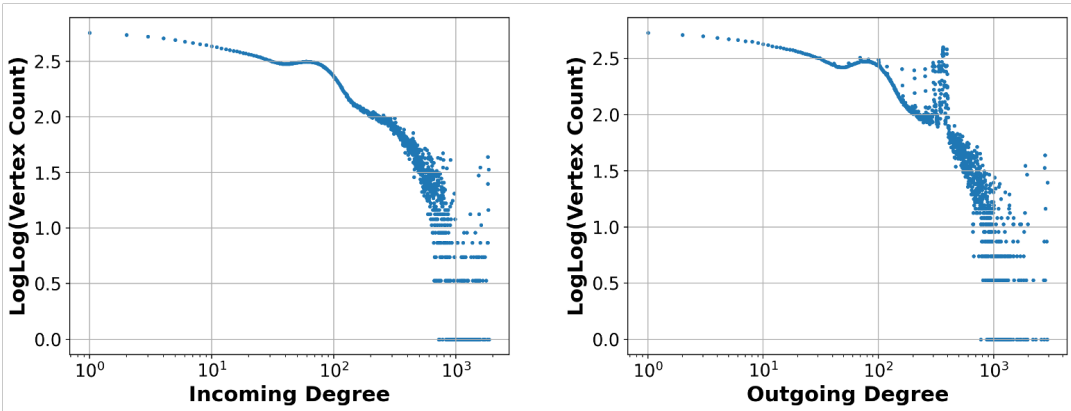


Figure 11 Vertex degrees of the overlap graph of metagenomic short reads (ERR3450203). The number of vertices per each degree is scaled as $\log \log y$. There is a clear peak around 10^2 at both incoming and outgoing degrees, which may imply dense existence of abundant species. The intermittent peaks after 10^2 might be related to repeating regions within and across species.

6 Discussion

To our knowledge, the Pro*k*rustean graph is the first model that elucidates the transitions of k-mers with respect to k sizes and serves as a framework for quantifying the influence of k sizes in applications. A new perspective on substring repeats is suggested: Think of sequences and maximal repeats ($\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$) as bins that disjointly contain k-mers of varying lengths co-occurring with them. Then, the behaviors of k-mer-based methods can be interpreted using these bins as representatives of all substrings. We posit that this principle will provide an entry point to design quantities that directly connect k-mer similarities to behaviors observed in applications, benefiting both practitioners and method developers.

The immediate application of the Pro*k*rustean graph is to compute k-mer-related quantities for a range of k values. We introduced various applications, including counting k-mers, calculating Bray-Curtis dissimilarity, and identifying maximal unitigs, where the computation was equivalent to scanning the graph a single time. This enables the analysis of k in diverse scenarios, e.g., maximizing the N50 of an assembly with respect to k, or computing Delta (Section 5.3).

Potential next steps include the implementation of varying-k methods. Stratifying regions reveal local patterns of repeats, suggesting that more advanced methods, such as de Bruijn graphs with locally modified k sizes, should be available. This approach differs from existing variable-order de Bruijn graphs [15], which maintain a large information space by traversing through different orders. The need for this so-called flexible approach stems from a lack of understanding of "locally optimal k" and faces scaling challenges. Therefore, a competitive method should use spaces that are comparable to a de Bruijn graph of a single order. Additionally, this practice can be enhanced by incorporating overlap information stored in the Pro*k*rustean graph, a hybrid approach that has been under discussion for some time [31].

With a broader view, the Pro*k*rustean graph of a genome can function as a reference database supporting sample querying and genome alignments as many aligners begin with a k-mer matching step where larger k can give better precision but lowers 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. Based on Section 5, it is not difficult to see 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 and affix tree - the union of suffix tree and reversed suffix tree - can easily extract each other.

In terms of practical applications, there is potential for enhancing the storage and construction 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. Furthermore, during the construction, 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 [18], adopting more advanced methods like those described in [17] could notably enhance performance.

The name of the Pro*k*rustean graph is inspired by Greek mythology: the bed of Procrustes, where guests were forced to fit his bed by stretching them or cutting off their limbs. Genomic sequences (guests) face even worse fates; they are chopped into k-mers, unaware of the extent of their information loss. However, the Pro*k*rustean graph (bed) now offers comfort to them, even though they will inevitably be sliced anyway.

References

1. Jarno N Alanko, Simon J Puglisi, and Jaakko Vuohtoniemi. Small searchable κ-spectra via subset rank queries on the spectral burrows-wheeler transform. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA23)*, pages 225–236. SIAM, 2023.
2. Hussah N AlEisa, Safwat Hamad, and Ahmed Elhadad. K-mer spectrum-based error correction algorithm for next-generation sequencing data. *Computational Intelligence and Neuroscience*, 2022, 2022.
3. Marleen Balvert, Xiao Luo, Ernestina Hauptfeld, Alexander Schönhuth, and Bas E Dutilh. Ogre: overlap graph-based metagenomic read clustering. *Bioinformatics*, 37(7):905–912, 2021.
4. Djamal Belazzougui and Fabio 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.
5. Gaëtan Benoit, Pierre Peterlongo, Mahendra Mariadassou, Erwan Drezen, Sophie Schbath, Dominique Lavenier, and Claire Lemaitre. Multiple comparative metagenomics using multiset k-mer counting. *PeerJ Computer Science*, 2:e94, 2016.
6. Vincenzo Bonnici and Vincenzo Manca. Informational laws of genome structures. *Scientific reports*, 6(1):28840, 2016.
7. Jessica K Bonnie, Omar Ahmed, and Ben Langmead. Dandd: efficient measurement of sequence growth and similarity. *bioRxiv*, pages 2023–02, 2023.
8. Florian P Breitwieser, Daniel N Baker, and Steven L Salzberg. Krakenuniq: confident and fast metagenomics classification using unique k-mer counts. *Genome biology*, 19:1–10, 2018.
9. Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de bruijn graphs. *Genome biology*, 22:1–24, 2021.
10. Yuval Bussi, Ruti Kapon, and Ziv Reich. Large-scale k-mer-based analysis of the informational properties of genomes, comparative genomics and taxonomy. *PloS one*, 16(10):e0258693, 2021.
11. Margherita Cavattoni and Matteo Comin. Classgraph: improving metagenomic read classification with overlap graphs. *Journal of Computational Biology*, 30(6):633–647, 2023.
12. Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014.
13. Illyoung Choi, Alise J Ponsero, Matthew Bomhoff, Ken Youens-Clark, John H Hartman, and Bonnie L Hurwitz. Libra: scalable k-mer–based tool for massive all-vs-all metagenome comparisons. *GigaScience*, 8(2):giy165, 2019.
14. Andrea Cracco and Alexandru I Tomescu. Extremely fast construction and querying of compacted and colored de bruijn graphs with ggcat. *Genome Research*, pages gr–277615, 2023.
15. Diego D'ıaz-Dom'ınguez, Taku Onodera, Simon J Puglisi, and Leena Salmela. Genome assembly with variable order de bruijn graphs. *bioRxiv*, pages 2022–09, 2022.
16. Veronika B Dubinkina, Dmitry S Ischenko, Vladimir I Ulyantsev, Alexander V Tyakht, and Dmitry G Alexeev. Assessment of k-mer spectrum applicability for metagenomic dissimilarity analysis. *BMC bioinformatics*, 17:1–11, 2016.
17. Johannes Fischer, Florian Kurpicz, and Marvin Löbel. Simple, fast and lightweight parallel wavelet tree construction. In *2018 Proceedings of ALENEX*, pages 9–20. SIAM, 2018.
18. Simon Gog, Timo Beller, Alistair Moffat, and Matthias 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.
19. Dan Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.
20. Rashedul Islam, Rajan Saha Raju, Nazia Tasnim, Istiak Hossain Shihab, Maruf Ahmed Bhuiyan, Yusha Araf, and Tofazzal 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.

21 Kazushi Kitaya and Tetsuo 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.

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

23 Thomas Krannich, W Timothy J White, Sebastian Niehus, Guillaume Holley, Bjarni V Halldórsson, and Birte Kehr. Population-scale detection of non-reference sequence variants using colored de bruijn graphs. *Bioinformatics*, 38(3):604–611, 2022.

24 Xingyu Liao, Min Li, Junwei Luo, You Zou, Fang-Xiang Wu, Yi Pan, Feng Luo, and Jianxin Wang. Improving de novo assembly based on read classification. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 17(1):177–188, 2018.

25 Vijini Mallawaarachchi. *Metagenomics Binning Using Assembly Graphs*. PhD thesis, The Australian National University (Australia), 2022.

26 Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, and Pavel A Pevzner. metaspades: a new versatile metagenomic assembler. *Genome research*, 27(5):824–834, 2017.

27 Ana Elena Pérez-Cobas, Laura Gomez-Valero, and Carmen Buchrieser. Metagenomic approaches in microbial ecology: an update on whole-genome and marker gene sequencing analyses. *Microbial genomics*, 6(8):e000409, 2020.

28 Alise Jany Ponsero, Matthew Miller, and Bonnie Louise Hurwitz. Comparison of k-mer-based de novo comparative metagenomic tools and approaches. *Microbiome Research Reports*, 2(4), 2023.

29 Andrey Prjibelski, Dmitry Antipov, Dmitry Meleshko, Alla Lapidus, and Anton Korobeynikov. Using spades de novo assembler. *Current protocols in bioinformatics*, 70(1):e102, 2020.

30 T Rhyker Ranallo-Benavidez, Kamil S Jaron, and Michael C Schatz. Genomescope 2.0 and smudgeplot for reference-free profiling of polyploid genomes. *Nat. comm.*, 11(1):1432, 2020.

31 Raffaella Rizzi, Stefano Beretta, Murray Patterson, Yuri Pirola, Marco Previtali, Gianluca Della Vedova, and Paola 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.

32 Luis M Rodriguez-r and Konstantinos T Konstantinidis. Estimating coverage in metagenomic data sets and why it matters. *The ISME journal*, 8(11):2349–2351, 2014.

33 Basir Shariat, Narjes Sadat Movahedi, Hamidreza Chitsaz, and Christina Boucher. Hydavista: towards optimal guided selection of k-mer size for sequence assembly. *BMC genomics*, 15(10):1–8, 2014.

34 Deyou Tang, Yucheng Li, Daqiang Tan, Juan Fu, Yelei Tang, Jiabin Lin, Rong Zhao, Hongli Du, and Zhongming Zhao. Kcoss: an ultra-fast k-mer counter for assembled genome analysis. *Bioinformatics*, 38(4):933–940, 2022.

35 Anuradha Wickramarachchi and Yu Lin. Metagenomics binning of long reads using read-overlap graphs. In *RECOMB International Workshop on Comparative Genomics*, pages 260–278. Springer, 2022.

36 Zhenhua Yang, Hong Li, Yun Jia, Yan Zheng, Hu Meng, Tonglaga Bao, Xiaolong Li, and Liaofu Luo. Intrinsic laws of k-mer spectra of genome sequences and evolution mechanism of genomes. *BMC Evolutionary Biology*, 20:1–15, 2020.

A Datasets

- Datasets mainly used in applications.
  SRR20044276, ERR3450203, SRR18495451, SRR21862404, SRR7130905
- Human pangenome references (Chromosome 1).
  AP023461.1, CH003448.1, CH003496.1, CM000462.1, CM001609.2, CM003683.2, CM009447.1, CM009872.1, CM010808.1, CM021568.2, CM034951.1, CM035659.1, CM039011.1, CM045155.1, CM073952.1, CM074009.1, CP139523.1, NC_000001.11, NC_060925.1
- Metagenome e.coli references.
  3682 E. coli assemblies in NCBI circa 2020. (https://zenodo.org/records/6577997)
- Metagenome sample for Bray-Curtis, referring to [28].
  ERS11976829, ERS11976830, ERS11976565, ERS11976566

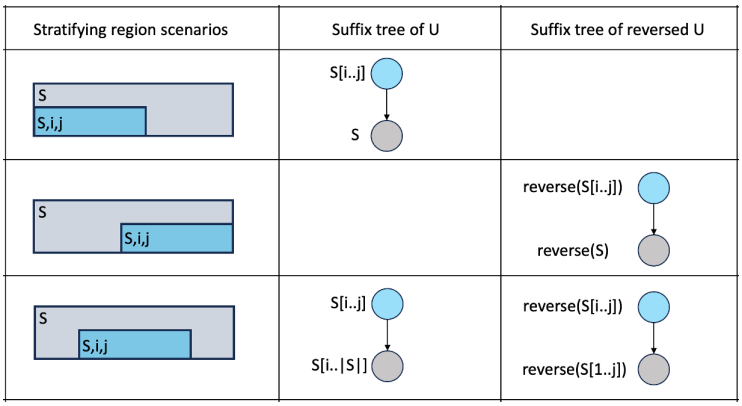
B Pro*k*rustean graph construction

Here, we briefly describe how the Pro*k*rustean graph is extracted from two generalized suffix trees - one built with $\mathcal{U}$ and the other with reverse strings of $\mathcal{U}$ - or an affix tree of $\mathcal{U}$.

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. Then below facts are straightforward.

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

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.



The above figure shows that every stratifying region corresponds to edge(s) of the suffix trees. For example, in row 1, since (S, i, j) can be extended to the right and its string is a maximal repeat, there exists a $node(str(S, i, j))$ with an edge branching to $S[j + 1]$.

However, the reverse does not hold, meaning an additional condition is required to determine which edges of the suffix trees imply stratifying relationships. For instance, even if the scenario of the first row 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, requiring further theorems.

Building even a single suffix tree is expensive in practice. The consequential algorithms of Pro*k*rustean graph construction use the eBWT input that implicitly navigate the suffix tree structure of $\mathcal{U}$. Further developments will be published in an extended version or in a different venue.

C Additional Proofs

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

▶ **Lemma 16.** *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). ◀

▶ **Proposition 17** (Complete). *$\mathcal{R}_{\mathcal{U}}$ is the set of maximal repeats of $\mathcal{U}$, i.e. a string R is captured from the construction of $\mathcal{R}_{\mathcal{U}}$ if and only if R is a maximal repeat.*

Proof. The forward direction is trivial that any string captured by a stratifying region is a maximal repeat by the assertion of decreasing occurrences.

The backward direction: Let R is a maximal repeat and consider its occurrence (S, i, j). By Lemma 16, (S, i, j) is either stratifying or has its extension stratifying. The former satisfies the condition asked by the proof, so check the latter. Let R_1 be the string of the extension of (S, i, j). Then R is a substring of R_1, so call an occurrence (R_1, i_1, j_1). Then again, (R_1, i_1, j_1) is either stratifying or has its extension stratifying. The process through $R_2, R_3, \ldots$ will eventually terminate at R_t as the lengths of captured maximal repeats decrease. Therefore, some occurrence (R_t, i_t, j_t) whose string is R is stratifying. ◀

▶ **Proposition 18** (Compact). *The number of incoming edges per vertex is at most $2|\Sigma|$, so*

$$|\mathcal{E}_{\mathcal{U}}| \leq 2|\Sigma||\mathcal{V}_{\mathcal{U}}|.$$

Proof. The inequality follows 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 maximal repeat. Let $StraRgn$ be the set of all stratifying regions represented by $\mathcal{E}_{\mathcal{U}}$. That is, $\forall c \in \Sigma$, $\forall R \in \mathcal{R}_{\mathcal{U}}$, $1 \geq |\{(S, i, j) | str(S, i, j) = R \ \& \ (S, i, j) \in StraRgn \ \& \ i > 1 \ \& \ S[i - 1] = c\}|$. Indeed, if this holds, the case of each letter at the immediate right of a region holds symmetrically, implying that $|\mathcal{E}_{\mathcal{U}}| \leq 2|\Sigma||\mathcal{V}_{\mathcal{U}}|$ as desired.

We argue by contradiction. For a given $c \in \Sigma$, let $(S, i, j) \in StraRgn$ has c at its immediate left: $S[i - 1] = c$. Assume for contradiction that there exists another stratifying region $(S', i', j') \in StraRgn$ 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. ◀

▶ **Proposition 19.** *Define $\mathcal{S}_{\mathcal{U}} :=$ a collection of co-occurring stacks that express all k-co-occurring regions found in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ each exactly once. Then,*

$$|\mathcal{S}_{\mathcal{U}}| \leq |\mathcal{V}_{\mathcal{U}}| + 2|\mathcal{E}_{\mathcal{U}}|.$$

Proof. Define a description of overlaps to pursue rigor.

▶ **Definition 20.** *Two regions are <u>k-conjoin</u> to each other if they overlap by $k - 1$ and does not include each other.*

(1) A k-co-occurring region and a k'-co-occurring region in S can be combined into the same *co-occurring stack* if they each k-conjoins to and k'-conjoins to the same stratifying region(s). Assuming without loss of generality that $k < k'$, and letting the k-co-occurring region be (S, i, j), then $(S, i, j, k_L = k, k_R = k')$ forms a valid *co-occurring stack*. Assume it is not a *co-occurring stack*: suppose there exists some region (S, i_k, j_k), defined by the stack, that is not k-co-occurring, implying (S, i_k, j_k) is included in a stratifying region. Then, (S, i_{k-1}, j_{k-1}) (which is a subset of (S, i_k, j_k)) is also contained in a stratifying region, and thereby inductively, contradicting the assertion that (S, i, j) is a k_L-co-occurring region.

(2) We aim to **maximally** capture such homogeneous regions in a *stack*. Then we can classify *stacks* by Types 0-2. Assuming all *maximal* Type 0/1/2 stacks are collected, then each k-co-occurring region of all k sizes has to be included in exactly one *co-occurring stack*. Then all substrings of all lengths are also each included in exactly one region represented by a *co-occurring stack*.

(3) A *maximal* Type 0 stack contains all whole regions, occurring $|\mathcal{V}_{\mathcal{U}}|$ times. A Type 0 stack takes the form $(S, 1, |S|, k_L = l_{max} + 1, k_R = |S|)$, where l_{max} is the length of the maximal stratifying region.

(4) A *maximal* Type 1 stack has the largest region that is k_R-co-occurring. The region conjoins to a stratifying region of length k_R. This is because regions in a Type 1 stack overlap with one stratifying region (of length l) on either their left or right and meet an edge on the opposite side, having the form as $(S, 1, *, k_L, k_R)$ or $(S, *, |S|, k_L, k_R)$. Therefore, the largest k-co-occurring region in a Type 1 stack should overlap with the stratifying region by l. So, $k_R = l$.

Similarly, a *maximal* Type 2 stack maintains the same relationship with at least one overlapping stratifying region. This argument follows the previous one, noting that the smaller stratifying region is of length k_R.

(5) Thus, for each Type 1 and 2 *co-occurring stack*, at least one stratifying region conjoining with its region should be of length k_R. Inversely, for each stratifying region of length l, there can be at most one *co-occurring stack* including the largest region be l-co-occurring on each side. Consequently, the number of Type 1 and 2 stacks cannot exceed twice the number of stratifying regions, i.e., $2|\mathcal{E}_\mathcal{U}|$.

Therefore, the collection of *co-occurring stacks* covering all k-co-occurring regions of all k sizes is of size at most $|\mathcal{V}_\mathcal{U}| + 2|\mathcal{E}_\mathcal{U}|$. ◀

D Additional Algorithms

D.1 Computing co-occurring stacks $\mathcal{S}_\mathcal{U}$

Recall the definition of $\mathcal{S}_\mathcal{U}$ and the proposition below.

▶ **Proposition 21.** *Define $\mathcal{S}_\mathcal{U}$:= a collection of co-occurring stacks that express all k-co-occurring regions found in $\mathcal{U} \cup \mathcal{R}_\mathcal{U}$ each exactly once. Then,*

$$|\mathcal{S}_\mathcal{U}| \leq |\mathcal{V}_\mathcal{U}| + 2|\mathcal{E}_\mathcal{U}|.$$

The goal is to construct $\mathcal{S}_\mathcal{U}$, i.e. cover all k-co-occurring regions with *co-occurring stacks* in time $O(|\text{Pro}k\text{rustean}|)$. The proof Appendix C showed that all *co-occurring stacks* can be classified by Type 0-2, and the number is smallest when considering maximal cases.

Algorithm 5 Computing maximal co-occurring stacks

Input: The Pro$k$rustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$
Output: Co-occurring stacks in v_S

```
Set S_U = {}
for v_S ∈ V_U do
    if type0Exists(S) then
        S_U ← S_U ∪ {type0(S)}
    for (S, i, j) s.t. type1LeftExists(S, i, j) do
        S_U ← S_U ∪ {type1Left(S, i, j)}
    for (S, i, j) s.t. type1RightExists(S, i, j) do
        S_U ← S_U ∪ {type1Right(S, i, j)}
    for (S, i, j, i', j') s.t. type2Exists(S, i, j, i', j') do
        S_U ← S_U ∪ {type2(S, i, j, i', j')}
return S_U
```

Type 0: no stratifying region is conjoining to the regions in the *stack*, so k_L is larger than the largest stratifying region.

$$\textit{type}0\textit{Exists}(S) := \text{no stratifying region is of length } |S| - 1.$$

$$\textit{type}0(S) := (S, 1, |S|, k_L = l_{max}(S) + 1, k_R = |S|), \text{ where}$$

$$l_{max}(S) := \text{the length of the largest stratifying region of } S.$$

Type 1: consider a Type 1 stack on the left side of the given stratifying region. Then k_L is longer than the maximum stratifying region on left of (S, i, j).

$\textit{type}1\textit{LeftExists}(S, i, j)$:= no stratifying region on left of (S, i, j) is $\geq |(S, i, j)|$.

$\textit{type}1\textit{Left}(S, i, j)$:=$(S, 1, i{+}k_L, k_L = |l_{Lmax}(S, i, j)|{+}1, k_R = |(S, i, j)|)$ where $l_{Lmax}(S, i, j)$:= maximum length among stratifying regions located on left of (S, i, j).

**Symmetrically define $l_{Rmax}(S, i, j)$, $type1LeftExists(S, i, j)$, $type1Left(S, i, j)$.

Type 2: there can be at most one Type 2 stack between two stratifying regions (S, i, j) and (S, i', j'). We should find the smallest k_L-co-occurring region that k_L-conjoins to both regions.

$type2Exists(S, i, j, i', j')$:= stratifying regions in the middle are $< min(|(S, i, j)|, |(S, i', j')|)$ and $|(S, i, j) \cap (S, i', j')| < k - 1$.

$type2(S, i, j, i', j') := (S, i_B, j_B, k_L = l_{Bmax}(S, i, j, i', j')+1, k_R = min(|(S, i, j)|, |(S, i', j')|)$ where $l_{Bmax}(S, i, j, i', j') :=$ maximum length of stratifying regions between (S, i, j) and (S, i', j'), and $i_B := j - k_L - |(i, j) \cap (i', j')|$, $j_B := i' + k_L + |(i, j) \cap (i', j')|$.

D.1.1 Complexity

The components of Type 0 and 1 are straightforward to compute in $O(|\text{Pro}k\text{rustean}|)$ time by traversing the stratifying regions once. However, finding all possible pairs satisfying $type2Exists(S, i, j, i', j')$ requires $O(|\mathcal{S}_{\mathcal{U}}|^2)$ time with a naive computation. We reduce it into a simpler problem and solve it.

Problem: Given an ordered set of intervals, find all pairs $((i, j), (i', j'))$ such that no interval in the middle is larger or equal to $min(|(i, j)|, |(i', j')|)$.

The number of valid pairs is upper bounded by 2* (the number of given intervals). It is because for a valid pair $((i, j), (i', j'))$, if $|(i, j)| \leq |(i', j')|$, then (i, j) does not appear on left in another valid pair. If such case exists for contradiction, i.e. $((i, j), (i'', j''))$ is valid and $|(i, j)| \leq |(i'', j'')|$, then any case of $i' < i''$ or $i'' < i'$ does not allow both pairs to be valid.

Algorithm 6 Find valid pairs

Input: Ordered set of intervals I
Output: Set of valid pairs of intervals

```
1  Initialize an empty stack S
2  Initialize an empty set of valid pairs P
3  for each interval (i_t, j_t) in I do
4      while S is not empty and |S.top()| ≤ |(i_t, j_t)| do
5          Add ((i_t, j_t), S.top()) to P
6          S.pop()
7      S.push(i_t, j_t)
8  for each interval (i_t, j_t) in I by reverse order do
9      while S is not empty and |S.top()| < (i_t, j_t) do
10         Add ((i_t, j_t), S.top()) to P
11         S.pop()
12     S.push(i_t, j_t)
13 return P
```

Correctness: Let $((i, j), (i', j'))$ be a valid pair and $|(i, j) \leq (i', j')|$. Consider line 3-7. Let intervals $(i_1, j_1), (i_2, j_2), \ldots, (i_n, j_n)$ are in the middle of the pair and are smaller than both (i, j) and (i', j') by definition. Then the line 4 must pop all the intervals after (i, j) because they are smaller than (i', j'). Also, (i, j) must be left in the stack, so $((i, j), (i', j'))$ is included in the output.

Symmetrically, if $|(i, j) > (i', j')|$, consider line 8-12.