bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Prokrustean Graph: A substring index for rapid k-mer size analysis

Adam Park[a], David Koslicki[a,b,c]

[a]Computer Science and Engineering in Pennsylvania State University, PA, USA
[b]Biology in Pennsylvania State University, PA, USA
[c]Huck Institutes of the Life Sciences in Pennsylvania State University, PA, USA

Abstract

Despite the widespread adoption of k-mer-based methods in bioinformatics, understanding the influence of k-mer sizes remains a persistent challenge. Selecting an optimal k-mer size or employing multiple k-mer sizes is often arbitrary, application-specific, and fraught with computational complexities. Typically, the influence of k-mer size is obscured by the outputs of complex bioinformatics tasks, such as genome analysis, comparison, assembly, alignment, and error correction. However, it is frequently overlooked that every method is built above a well-defined k-mer-based object like Jaccard Similarity, de Bruijn graphs, k-mer spectra, and Bray-Curtis Dissimilarity. Despite these objects offering a clearer perspective on the role of k-mer sizes, the dynamics of k-mer-based objects with respect to k-mer sizes remain surprisingly elusive.

This paper introduces a computational framework that generalizes the transition of k-mer-based objects across k-mer sizes, utilizing a novel substring index, the Pro*k*rustean graph. The primary contribution of this framework is to compute quantities associated with k-mer-based objects for all k-mer sizes, where the computational complexity depends solely on the number of maximal repeats and is independent of the range of k-mer sizes. For example, counting vertices of compacted de Bruijn graphs for $k = 1, \ldots, 100$ can be accomplished in mere seconds with our substring index constructed on a gigabase-sized read set.

Additionally, we derive a space-efficient algorithm to extract the Pro*k*rustean graph from the Burrows-Wheeler Transform. It becomes evident that modern substring indices, mostly based on longest common prefixes of suffix arrays, inherently face difficulties at exploring varying k-mer sizes due to their limitations at grouping co-occurring substrings.

We have implemented four applications that utilize quantities critical in modern pangenomics and metagenomics. The code for these applications and the construction algorithm is available at `https://github.com/KoslickiLab/prokrustean`.

Keywords: k-mer, k-mer spectra, FM-index, BWT, genome assembly, pangenomics, metagenomics
2000 MSC: 03B70, 05C85, 92-08

1. Introduction

As the volume and number of sequencing reads and reference genomes grow every year, k-mer-based methods continue to gain popularity in computational biology. This simple approach—cutting sequences into substrings of a fixed length—offers significant benefits across various disciplines. Biologists consider k-mers as intuitive markers representing biologically significant patterns, bioinformaticians easily formulate novel methods by regarding k-mers as fundamental units that reflect their original sequences, and engineers leverage their fixed-length nature to optimize computational processes at a very low level. However, the understanding of the most crucial parameter, k, remains surprisingly elusive, thereby impeding further methodological advancements.

Two central challenges emerge in the application of k-mer-based methods. First, the selection of the k-mer size is often arbitrary, despite its well-recognized influence on outcomes. This issue, though widely acknowledged, remains insufficiently addressed in the literature, with little formal guidance on how to determine an optimal k size for different applications. The reasoning behind these choices is frequently obscured, typically confined to specific, unpublished experimental analyses (e.g. "we found that $k = 31$ was appropriate..."). Second, methods attempting to utilize multiple

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

k-mer sizes encounter significant computational burdens. While incorporating multiple k-mer sizes can naturally enhance the accuracy of k-mer-based methods, the computational costs escalate with each additional k-mer size used. Consequently, researchers often resort to "folklore" k value(s) based on prior empirical results.

The influence of k-mer sizes within each method is a complex function reflecting bioinformatics pipelines that process k-mers. As biological adjustments and engineering strategies complicate the pipelines, their outputs obscure the impact of k-mer sizes with noisy factors, as simplified in the following abstraction:

$$\texttt{pipeline}(k\texttt{-mer-based objects}(\textit{sequences}, k),\ \texttt{biological adjustments},\ \texttt{engineering}).$$

Despite the complexity of the pipelines, there always exist mathematically well-defined k-mer-based objects that form the foundation of method formulation. These objects abstract the utilization of k-mers and depend solely on sequences and k-mer sizes. Thus, they offer potential for generalizability and quantification of the influence of k-mers in methods. Indeed, intuitive quantities have been derived from k-mer-based objects; however, there are challenges in actually computing them, as detailed in several examples we now present.

In genome analysis, the number of distinct k-mers is often used to reflect the complexity of genomes. Although the number varies by k-mer sizes, large data sizes restrict experiments to a few k values [9, 39, 15]. A recent study suggests that the number of distinct k-mers across all k-mer sizes provides additional insights into pangenome complexity [10]. Furthermore, the frequencies of k-mers provide richer information, as discussed in [1, 46], but their computation becomes more complicated and sometimes impractical, even with a fixed k-mer size [7].

In comparative analyses, Jaccard Similarity is utilized for genome indexing and searching by being approximated through hashing techniques [34, 26]. Experiments attempting to assess the influence of k-mer sizes on Jaccard Similarity must undergo tedious iterations through various k-mer sizes [8]. Additionally, Bray-Curtis dissimilarity is a k-mer frequency-based metric frequently used in comparing metagenomic samples, where experiments face resource limitations due to the growing size of sequencing data, even with a fixed k-mer size. Yet, the demand for analyzing multiple k-mer sizes continues to increase [23, 36, 35].

Genome assemblers utilizing k-mer-based de Bruijn graphs are probably the most sensitive to k-mer sizes, and those employing multi-k approaches face significant computational challenges. The choice of k-mer sizes is particularly crucial in de novo assemblers, yet it predominantly relies on heuristic methods [27, 19, 38]. The topological features of de Bruijn graphs are succinctly summarized by the compacting process, where vertices represent simple paths called unitigs, but exploring these features across varying k-mer sizes is computationally intensive. Furthermore, the development of multi-k de Bruijn graphs continues to be a challenging and largely theoretical endeavor [40, 43, 22], with no substantial advancements following the heuristic selection of multiple k-mer sizes implemented by metaSPAdes [33].

These examples motivate the pressing need to generalize the exploration of k-mer-based objects across k-mer sizes. Our manuscript introduces a framework for rapidly computing quantities derived from k-mer-based objects, thereby addressing the prevalent challenge in bioinformatics. The organization of the manuscript is as follows:

- Section 2 defines the main objective of computing k-mer-based quantities and introduces the proxy problem of computing substring co-occurrence.
- Section 3 defines $\mathcal{G}_\mathcal{U}$, the Pro*k*rustean graph, a novel substring representation that provides straightforward access to substring co-occurrence.
- Section 4 outlines a framework built upon the Pro*k*rustean graph, accompanied by algorithms that treat various k-mer-based objects and compute related quantities across all possible k-mer sizes in $O(|\mathcal{G}_\mathcal{U}|)$ time.
- Section 5 presents the experimental results of computing k-mer-based quantities.
- Section 6 derives the construction algorithm of $\mathcal{G}_\mathcal{U}$, and discusses the limitations of other modern substring indices in computing substring co-occurrence.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

2. Problem Formulation

2.1. Basic Notations

Let Σ be our alphabet and let $S \in \Sigma^*$ represent a finite-length string, and $\mathcal{U} \subset \Sigma^*$ for a set of strings. Because of the biological sequence motivation for this work, the elements of $\mathcal{U}$ are called *sequences*. Consider the following preliminary definitions:

Definition 1.
- *A region in the string S is a triple (S, i, j) such that $1 \le i \le j \le |S|$.*
- *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):*

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

- *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.*
- *$\mathcal{R}_{\mathcal{U}}$ is the set of all maximal repeats in $\mathcal{U}$.*

2.2. The proxy problem: how to compute substring co-occurrence?

There exists a theoretical void in identifying when a k-mer and a k'-mer serve similar roles within their respective substring sets. Although close k-mer sizes generally yield comparable outputs, dissecting this phenomenon at the level of local substring scopes is unexpectedly challenging. For instance, de Bruijn graphs constructed from sequencing reads with $k = 30$ and $k' = 31$ display distinct yet highly similar topologies [40]. However, anyone formally articulating the topological similarity would find it quite elusive, as vertex mapping and other techniques establishing correspondences between the two graphs fail to consistently explain it.

A common underlying difficulty is that the entire k-mer set undergoes complete transformations as the k-mer size changes. Since the role of a k-mer is assigned within its k-mer set, the role of a k'-mer within k'-mers cannot be "locally" derived. To address this issue, we propose substring co-occurrence as a generalized framework for consistently grouping k-mers of similar roles across varying k-mer sizes.

Definition 2. *A string S co-occurs within a string S' in $\mathcal{U}$ if:*

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

Modern substring indexes built on longest common prefixes (LCP) of their (implicit) suffix array are adept at capturing co-occurrence in this "extending direction." In suffix trees, a substring that extends from S to S' along an edge—without passing through a node—indicates preservation of co-occurrence. Similarly, in the Burrows-Wheeler Transform (BWT) and its variants such as the FM-index and r-index, representing both S and S' with the same set of LCPs corresponding to some suffix array interval implies co-occurrence. However, there has been no recognized necessity for addressing the following "substring direction":

Definition 3. *$co\text{-}substr_{\mathcal{U}}(S)$ is the set of substrings that co-occur within S.*

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Modern substring indexes are not efficient at computing $co\text{-}substr_{\mathcal{U}}(S)$, which results in their failure to "smoothly" explore k-mer-based objects across varying k-mer sizes. Specifically, shrinking a substring found in $\mathcal{U}$ always requires some link (or pointer) to navigate to the corresponding LCP in the (implicit) suffix array. Given that the number of substrings scales quadratically with the cumulative size of the input sequences, this linkage system becomes impractical. For instance, the variable-order de Bruijn graph and its variants address the space issue by employing a compact de Bruijn graph representation built above the Burrow-Wheeler transform [12]; however, this solution requires non-trivial operation times for both "forward" moves and changes in order (k) [11, 5]. So, these data structures do not allow extensive exploration of the substring space, and hence their multi-k approaches do not scale well.

Our work demonstrates that $co\text{-}substr_{\mathcal{U}}(S)$ can be computed efficiently, and then the functionality can be used to rapidly compute k-mer quantities. We introduce a simple yet foundational property as groundwork.

Theorem 1 (Principle)**.** *For any substring S such that $occ_{\mathcal{U}}(S) > 0$,*

$$S \textit{ co-occurs within exactly one sequence or maximal repeat in } \mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$$

Proof. Assume the theorem does not hold, i.e., S co-occurs within either no string or multiple strings in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. Consider the former case: S co-occurs within no string in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$, meaning S occurs more than any of its superstrings in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. Given that S occurs in some string $S' \in \mathcal{U}$ ($i.e. occ_{\mathcal{U}}(S) > 0$), S must be occurring more frequently than S' by assumption. Then, there exists a maximal repeat R such that S is a substring of R and R is a substring of S', because extending S within S' decreases its occurrence at some point. Again, whenever S is a substring of a maximal repeat R, S must be occurring more frequently than R by assumption. Following the similar argument above, there exists another maximal repeat R' such that S is a substring of R' and R' is a proper substring of R. This recursive argument will eventually terminate as R' is strictly shorter than R. Thus, S co-occurs within the last maximal repeat, which contradicts the assumption.

Now consider the latter case where S co-occurs within at least two strings in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. If S is extended within these two strings, either to the right or left by one step at each time, the extensions eventually diverge into two different strings since the two superstrings differ, thereby reducing the number of occurrences. Hence, S must be occurring more frequently than at least one of those two superstrings, leading to a contradiction. □

This theorem provides two key insights. First, every substring found in $\mathcal{U}$ exclusively co-occurs as a substring of either a sequence or a maximal repeat in $\mathcal{U}$, so the collective sets of $co\text{-}substr_{\mathcal{U}}(S)$ from every $S \in \mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ comprehensively and disjointly cover the entire substring space. Second, by computing $co\text{-}substr_{\mathcal{U}}(S)$ for every $S \in \mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$, we can categorize k-mers of similar roles across various lengths, devising various algorithmic ideas on $co\text{-}substr_{\mathcal{U}}(S)$. The following main theorem outlines the components discussed in the remainder of the manuscript.

Theorem 2. ***(Main Result)*** *Given a set of sequences $\mathcal{U}$, there exists a substring representation $\mathcal{G}_{\mathcal{U}}$ of size $O(|\Sigma| \cdot |\mathcal{R}_{\mathcal{U}}|)$. $\mathcal{G}_{\mathcal{U}}$ can be used to compute $\underline{\text{k-mer quantities}}$ of $\mathcal{U}$ for all $k = 1, \ldots, k_{\max}$ within $O(|\mathcal{G}_{\mathcal{U}}|)$ time and space, where $k_{\max}$ is the longest sequence length in $\mathcal{U}$. The $\underline{\text{k-mer quantities}}$ are given as follows:*

- *(Counts) The number of distinct k-mers, as used in genome cardinality DandD [10].*
- *(Frequencies) k-mer Bray-Curtis dissimilarities, as used in comparing metagenomic samples [36].*
- *(Extensions) The number of unitigs in k-mer de Bruijn graphs, as used in analyses of reference genomes and genome assembly [29, 42].*
- *(Occurrences) The number of edges with annotated length of k in the overlap graph. [45].*

Furthermore, the graph $\mathcal{G}_{\mathcal{U}}$ can be constructed in $O(|BWT| + |\mathcal{G}_{\mathcal{U}}|)$ time and space, where $|BWT|$ is the size of the Burrows-Wheeler transform representation of $\mathcal{U}$.

Proof. Section 3 analyzes the size of $\mathcal{G}_{\mathcal{U}}$, Section 4 introduces the computation of k-mer quantities, and Section 6 derives the construction algorithm. □

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

The main contribution is that the time complexity for computing k-mer quantities for all $k \in [1, k_{\max}]$ using $\mathcal{G}_{\mathcal{U}}$ is independent of the range of k-mer sizes, which is $O(|\mathcal{G}_{\mathcal{U}}|)$. Specifically, letting N be the cumulative sequence length of $\mathcal{U}$, $|\mathcal{G}_{\mathcal{U}}|$ is sublinear relative to N. In contrast, the direct computation of k-mer quantities from the sequence set $\mathcal{U}$ for any fixed k-mer size demands, at best, $O(N)$ time. Extending this computation naively to cover all $k = 1, \ldots, k_{\max}$ would exponentially increase the computational demand to $O(N^{k_{\max}})$. This significant improvement in complexity leverages a proper representation of repeats in $\mathcal{U}$, as will be introduced in the next section.

3. Pro*k*rustean graph: A hierarchy of maximal repeats

The key idea of the Pro*k*rustean graph is to recursively capture maximal repeats by their relative frequencies of occurrences in $\mathcal{U}$. Consider the following two descriptions of repeats in three sequences: ACCCT, GACCC, and TCCCG.

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.

Each first and second case uses 5 and 4 units of occurrences, respectively, yet preserves the same meaning. As depicted in Figure 1, the rule of the compact second case is to recursively capture regions of more frequent substrings.

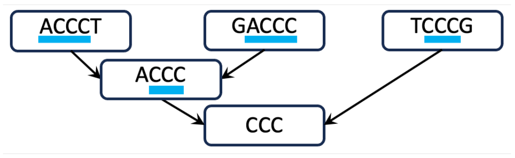


Figure 1: Recursive repeats. At each sequence on the top level, maximally long subregions (indicated with blue underlining) are used to denote substrings that are more frequent than its superstring. The arrows then points to a node of the substring, and the process repeats hierarchically.

Definition 4. *(S, i, j) is a locally-maximal repeat region 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),

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

***Note that we often omit "in $\mathcal{U}$" when the context is clear.*

A locally-maximal repeat region captures a substring that appears more frequently than the "parent" string, and any extension of the region captures a substring that appears as frequently as the parent string. Consider the previous example $\mathcal{U}$ = {ACCCT, GACCC, TCCCG}. (ACCCT, 1, 4) is a locally-maximal repeat region capturing the substring ACCC in ACCCT. However, (ACCCT, 2, 4) is not a locally-maximal repeat region because the region capturing CCC in ACCCT can be extended to the left to capture ACCC which occurs more frequently than ACCCT.

Note, an alternative definition using substring notations, such as *locally-maximal repeats*, instead of regions, is not robust. Consider $\mathcal{U}$ = {ACCCTCCG, GCCC} and S :=ACCCTCCG. Observe that the region $(S, 2, 4)$ capturing CCC is a locally-maximal repeat region because CCC occurs more frequently than S in $\mathcal{U}$, but its immediate left and right extensions capture ACCC and CCCT that occur the same number of times as S. In constrast, a subregion $(S, 2, 3)$ capturing CC is not a locally-maximal repeat region because an extension $(S, 2, 4)$ still captures a string (CCC) which occurs 2 times in $\mathcal{U}$ which is more than S. However, $(S, 6, 7)$, which also captures CC, *is* a locally-maximal repeat region because $|occ_{\mathcal{U}}(\text{CC})| = 5 > |occ_{\mathcal{U}}(S)| = 1$ and $|occ_{\mathcal{U}}(S)| = |occ_{\mathcal{U}}(\text{TCC})| = |occ_{\mathcal{U}}(\text{CCG})|$. Consequently, defining a *locally-maximal repeat* as $S_{6..7}$ =CC becomes ambiguous with $S_{2..3}$=CC, whereas an explicit expression of a region $(S, 6, 7)$ more accurately reflects the desired property of hierarchy of occurrences.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

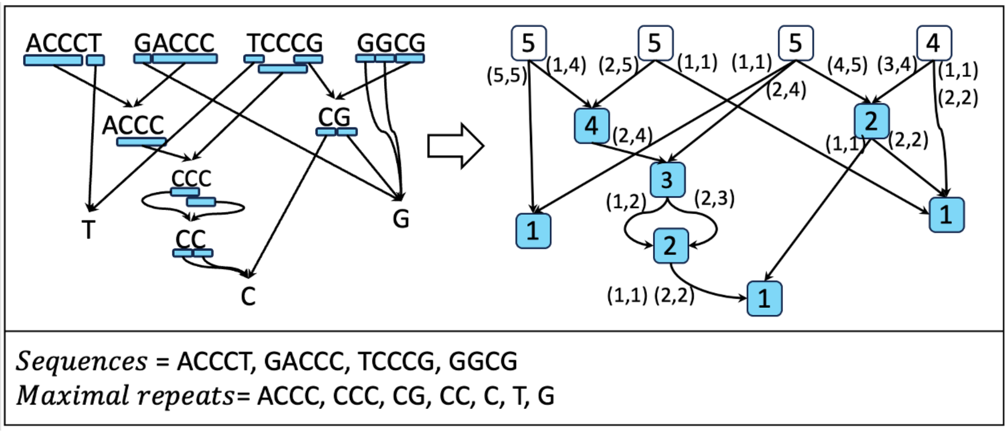


Figure 2: The Pro*k*rustean graph of $\mathcal{U}$={ACCCT,GACCC,TCCCG,GGCG}. The left graph describes the recursion of locally-maximal repeat regions, and the Pro*k*rustean graph on the right represents the same structure with integer labels storing the regions and the size of the substrings.

3.1. Prokrustean Graph

The Pro*k*rustean graph of $\mathcal{U}$ is simply a graph representation of recursive locally-maximal repeat regions on $\mathcal{U}$, i.e. vertexes represent substrings and edges represent locally-maximal repeat regions. The theorem below says that all maximal repeats in $\mathcal{U}$ are caught along the recursive description.

Theorem 3 (Complete)**.** *Construct a string set* $\mathbf{R}(\mathcal{U})$ *as follows:*

1. *for each locally-maximal repeat region* (S, i, j) *where* $S \in \mathcal{U}$*, add* $str(S, i, j)$ *to* $\mathbf{R}(\mathcal{U})$ *and*

2. *for each locally-maximal repeat region* (S, i, j) *where* $S \in \mathbf{R}(\mathcal{U})$*, add* $str(S, i, j)$ *to* $\mathbf{R}(\mathcal{U})$*.*

It follows that $\mathbf{R}(\mathcal{U}) = \mathcal{R}_{\mathcal{U}}$ *holds.*

Proof. Any string in $\mathbf{R}(\mathcal{U})$ is a maximal repeat of $\mathcal{U}$, so the claim is satisfied if the process captures every maximal repeat of $\mathcal{U}$. Assume a maximal repeat R is not in $\mathbf{R}(\mathcal{U})$. Consider any occurrence of R in some $S \in \mathcal{U}$. Extending the maximal repeat R within S makes the number of its occurrences drop, but since R cannot occur as a locally-maximal repeat region by assumption, it must be included in some locally-maximal repeat region that captures a maximal repeat R_1, hence $R_1 \in \mathbf{R}(\mathcal{U})$, and R is a proper substring of R_1. Now consider any occurrence of R within R_1. This argument continues recursively, capturing maximal repeats $R_1, R_2, ...$, but cannot extend indefinitely as R_{i+1} is always shorter than R_i for every $i \geq 1$. Eventually, R must be occurring as a locally-maximal repeat region of some R_n, which is a contradiction. □

Therefore, we use maximal repeats as vertices of the Pro*k*rustean graph, along with sequences, so $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$.

Definition 5. *The Prokrustean graph of* $\mathcal{U}$ *is a directed multigraph* $\mathcal{G}_{\mathcal{U}} := (\mathcal{V}_{\mathcal{U}}, \mathcal{E}_{\mathcal{U}})$*:*

- $\mathcal{V}_{\mathcal{U}}$*:* $v_S \in \mathcal{V}_{\mathcal{U}}$ *if and only if* $S \in \mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$*.*
- *Each vertex* $v_S \in \mathcal{V}_{\mathcal{U}}$ *is annotated with the string size,* $size(v_S) = |S|$*.*
- $\mathcal{E}_{\mathcal{U}}$*:* $e_{S,i,j} \in \mathcal{E}_{\mathcal{U}}$ *if and only if* (S, i, j) *is a locally-maximal repeat region.*
- *Each edge* $e_{S,i,j} \in \mathcal{E}_{\mathcal{U}}$ *directs from* v_S *to* $v_{str(S,i,j)}$ *and is annotated with* $interval(e_{S,i,j}) := (i, j)$*, to represent the locally-maximal repeat region.*

Figure 2 visualizes how locally-maximal repeat regions are encoded in a Pro*k*rustean graph. Next, the cardinality analysis of this representation reveals promising bounds.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Theorem 4 (Compact)**.** $|\mathcal{E}_\mathcal{U}| \le 2|\Sigma||\mathcal{V}_\mathcal{U}|$.

Proof. Every vertex in $\mathcal{V}_\mathcal{U}$ has at most one incoming edge per letter extension on the right (or similarly on the left), so has at most $2|\Sigma|$ incoming edges. Assume the contrary: a maximal repeat appears as two different locally-maximal repeat regions (S, i, j) and (S', i', j') within $S, S' \in \mathcal{U} \cup \mathcal{R}_\mathcal{U}$, and can extend by the same letter on the right, i.e., $(S, i, j+1)$ and $(S', i', j'+1)$ capture the same string. Extending these regions further together will eventually diverge their strings, because either $S \neq S'$ or the original two regions are differently located even if $S = S'$. Consequently, at least one of them—$(S, i, j+1)$ without loss of generality—results in a decreased occurrence count of its string as it extends. Hence, $|occ_\mathcal{U}(str(S, i, j+1))| > |occ_\mathcal{U}(S)|$. This leads to contradiction because a locally-maximal repeat region (S, i, j) should satisfy $|occ_\mathcal{U}(S, i, j+1)| = |occ_\mathcal{U}(S, 1, |S|)|$. □

Assuming trivially that $|\mathcal{U}| \ll |\mathcal{R}_\mathcal{U}|$, meaning there are significantly more maximal repeats than the number of sequences, we derive that $O(|\mathcal{V}_\mathcal{U}|) = O(|\mathcal{R}_\mathcal{U}|)$, hence $O(|\mathcal{G}_\mathcal{U}|) := O(|\mathcal{E}_\mathcal{U}|) = O(|\Sigma||\mathcal{R}_\mathcal{U}|)$ by the theorem. Given that $|\Sigma|$ is typically constant in genomic sequences (eg. $\Sigma = \{A, C, T, G\}$), the graph's size depends on the number of maximal repeats. Letting N be the accumulated sequence length of $\mathcal{U}$, it is a well-known fact that $|\mathcal{R}_\mathcal{U}| < N$ [25], so the graph grows sublinear to the input size. Furthermore, in the implementations described in Section 5, $\mathcal{R}_\mathcal{U}$ can be restricted to maximal repeats of length at least $k_{\min}$. Although the size of $\mathcal{G}_\mathcal{U}$ is comparable to that of the suffix tree of $\mathcal{U}$, setting $k_{\min}$ allows for significantly more efficient and configurable space usage.

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 $O(|\mathcal{G}_\mathcal{U}|)$ time and space.

4.1. Accessing co-occurring k-mers for a single k size

We briefly cover how k-mers of a fixed k-mer size are accessed through the Pro*k*rustean graph, before generalizing the computation to all k-mer sizes in the next section. Previously, a toy example in Figure 1 depicted blue regions that cover locally-maximal repeat regions. Then, given a 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 3.

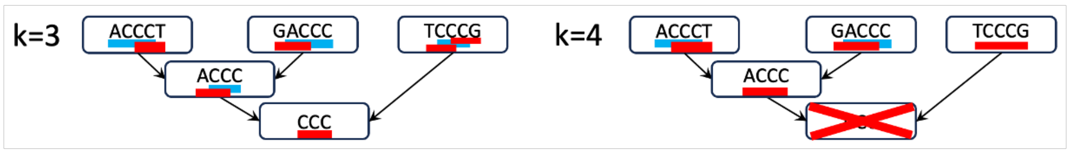


Figure 3: Accessing k-mers by complementing locally-maximal repeat regions. Blue underlining depicts locally-maximal regions, and red underlining depicts regions that complemented k-mers not covered in blue regions, given sizes of $k = 3$ and 4. Every k-mer appears exactly once within a red region. For example, at the figure of $k = 3$ on left, the 3-mer ACC is included in a red region in ACCCC and does not appear again in any other red region.

An opportunistic property is that k-mers in red regions co-occur within their parent strings. Following notation captures substrings in red regions:

Definition 6. *A string S k-co-occurs within a string S' in $\mathcal{U}$ if:*

$$\textit{every } k\textit{-mer in } S \textit{ co-occurs within } S' \textit{ in } \mathcal{U}.$$

We mean the same by S being a k-co-occuring substring of S'. Also, S maximally k-co-occurs within S' in $\mathcal{U}$ if no superstring of S k-co-occurs within S' in $\mathcal{U}$. Maximal k-co-occurring substrings of S' are easily computed as complementary regions in S' as introduced in Figure 3. So, they are efficiently identified with the Pro*k*rustean graph, as implied by the proposition below.

Proposition 1. *A string S maximally k-co-occurs within a string S' in U if and only if its region in S' intersect locally-maximal repeat regions by under $k-1$, and on both sides, either extends to an end of S' or intersects a locally-maximal repeat region by exactly $k-1$.*

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Proof. The forward direction is straightforward: if S k-co-occurs within S', its region cannot intersect any locally-maximal repeat region by k at any case, and if the intersection is under $k-1$ on its edge, by extending S and making the intersection $k-1$, another superstring of S k-co-occurs within S'.

For the reverse direction, assume a region satisfies the conditions. A k-mer appearing in the region cannot occur more than S, as it would mean the region intersects a locally-maximal repeat region by at least k, so S' k-co-occurs within S. S' is also maximal because extending its region increases the size of the intersection to more than $k-1$. □

Proposition 1 implies that computing maximal k-co-occurring substrings of a string S takes time linear to the number of locally-maximal repeat regions in S, given the regions pre-ordered by positions: Enumerate locally-maximal repeat regions of length at least k and check its left and right whether a complementary region of length at least k can be defined. Lastly, maximally capture complementary regions on both sides so that they either intersect locally-maximal repeat regions by $k-1$ or meet an end of S. See Figure 4 for a detailed example.

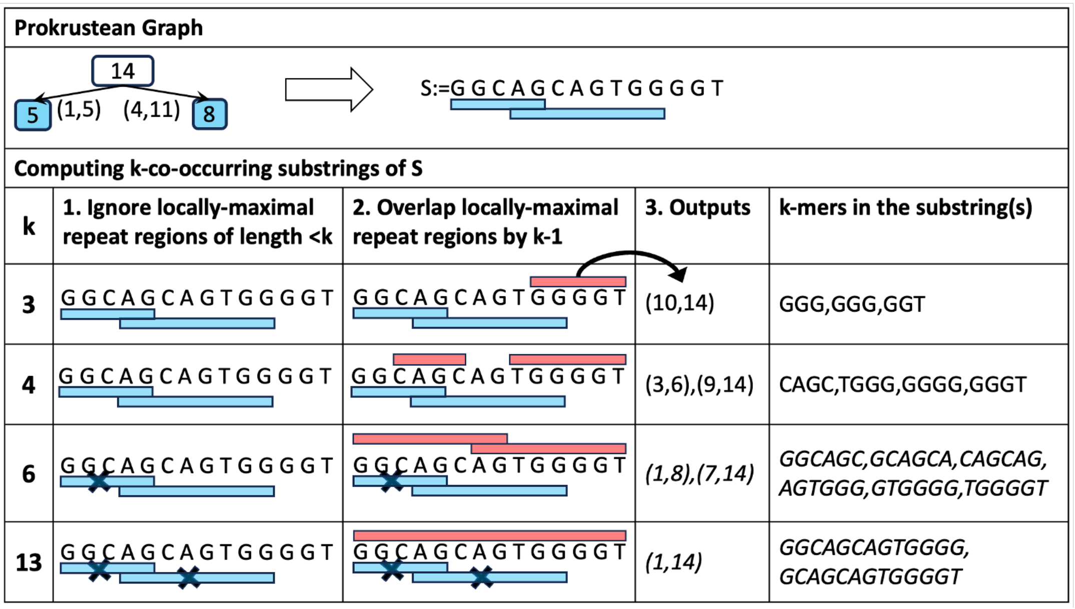

Prokrustean Graph
14
5
(1,5)
(4,11)
8
S:=GGCAGCAGTGGGGT
Computing k-co-occurring substrings of S
k
1. Ignore locally-maximal repeat regions of length <k
2. Overlap locally-maximal repeat regions by k-1
3. Outputs
k-mers in the substring(s)
3
GGCAGCAGTGGGGT
(10,14)
GGG,GGG,GGT
4
(3,6),(9,14)
CAGC,TGGG,GGGG,GGGT
6
(1,8),(7,14)
GGCAGC,GCAGCA,CAGCAG,AGTGGG,GTGGGG,TGGGGT
13
(1,14)
GGCAGCAGTGGGG,GCAGCAGTGGGGT

Figure 4: Computing k-co-occurring substrings (red) of S of varying k sizes. The rule is to cover every k-mer that is not covered by a locally-maximal repeat region (blue). Note that a k-co-occurring substring can appear on the intersection of two locally-maximal repeat regions too. For example, the region (S, 3,6) capturing CAGC in the second row ($k = 4$) overlaps two blue regions by 3.

Therefore, the idea for computing complementary regions can be applied to count distinct k-mers for a fixed k-mer size. The toy algorithm below takes $O(|\mathcal{G}_\mathcal{U}|)$ time and space.

```
Algorithm 1: Count distinct $k$-mers of a single $k$
    Input: A Prokrustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$ , a $k$ size
    Output: The number of distinct $k$-mers in $\mathcal{U}$
1   for $v_S \in \mathcal{V}_\mathcal{U}$ do
2       Compute complementary regions by outgoing edges of $v_S$.
3       for $(i, j)$ in complementary regions do
4           $count += j - i + 2 - k$
5   return count
```

Note that line 2 is the computation derived from Proposition 1, as depicted in Figure 4. The locally-maximal repeat regions of S are represented by the outgoing edges of v_S. The nested loop on lines 1 and 2 thus takes $O(|\mathcal{E}_\mathcal{U}|)$, i.e., $O(|\Sigma||\mathcal{R}_\mathcal{U}|)$ time. For correctness, Theorem 1 states that any k-mer K co-occurs within exactly one string S in

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

$\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. K extends to a unique maximal k-co-occuring substring of S; otherwise, multi-occurrence in S is implied. Lastly, Proposition 1 guarantees that every maximal k-co-occuring substring of S can be identified by scanning the outgoing edges of v_S.

4.2. *Accessing co-occurring k-mers for a range of k*

The computation can be extended to a range of k-mer sizes while preserving the complexity. Recall that the problem is to efficiently express *co-substr*$_{\mathcal{U}}(S)$. It turns out that *co-substr*$_{\mathcal{U}}(S)$ can be expressed by a finite number of stack-like substring structures.

Definition 7. *Consider two strings S and T where T is a substring of S, along with two numbers: a depth d^* and a k-mer size k^*, each at least 1. A co-occurrence stack of S in U is defined as a 4-tuple:*

$$(S, T, k^*, d^*)$$

$$\textit{when } str(T, 1 + \delta_l \cdot (i-1), |T| - \delta_r \cdot (i-1)) \textit{ maximally } k_i\textit{-co-occurs within } S$$

where $k_i = k^* - (i-1)$ for $i = 1, 2, \ldots, d^*$, with change rates $\delta_l := \mathbf{1}_{T \text{ is NOT a prefix of } S}$ and $\delta_r := \mathbf{1}_{T \text{ is NOT a suffix of } S}$.

A co-occurrence stack expresses a k-mer if it is a substring of some k-co-occurring substring identified by the stack. A co-occurrence stack is maximal if its expressed k-mers are not a subset of those expressed by any other co-occurrence stack. It is clear that any k-mer expressed in a co-occurrence stack of S co-occurs within S. So, maximal co-occurrence stacks of S collectively express *co-substr*$_{\mathcal{U}}(S)$.

Definition 8. *co-stack*$_{\mathcal{U}}$*(S) is the set of all maximal co-occurrence stacks of S in* $\mathcal{U}$.

A maximal co-occurrence stack of S is type-0 if the substring T is S itself, type-1 if T is a proper prefix or suffix of S, and type-2 otherwise. The numeral in each type's name indicates the number of locally-maximal repeat regions intersecting the regions of substrings identified by the co-occurrence stack. Refer to Figure 5 for intuition on the type names.

Theorem 5. *co-stack*$_{\mathcal{U}}(S)$ *completely and disjointly cover co-occurring substrings of S, i.e.,*

$$\textit{co-substr}_{\mathcal{U}}(S) = \bigsqcup_{(S,T,k^*,d^*) \in \textit{co-stack}_{\mathcal{U}}(S)} \{\textit{substrings expressed by } (S, T, k^*, d^*)\}.$$

Also,

$$|\textit{co-stack}_{\mathcal{U}}(S)| \leq 1 + 2 \cdot (\textit{the number of locally-maximal repeat regions in } S)$$

Proof. Appendix B.1 covers the proof. □

Lastly, this scheme is extended to the entire substring space. Also, all maximal co-occurrence stacks in $\mathcal{U}$ can be enumerated within the computation time $O(|\mathcal{E}_{\mathcal{U}}|)$.

Proposition 2. *The number of all maximal co-occurrence stacks in $\mathcal{U}$ is $O(|\mathcal{G}_{\mathcal{U}}|)$. Precisely,*

$$\sum_{S \in \mathcal{U} \cup \mathcal{R}_{\mathcal{U}}} |\textit{co-stack}_{\mathcal{U}}(S)| \leq |\mathcal{V}_{\mathcal{U}}| + 2|\mathcal{E}_{\mathcal{U}}|.$$

Proof. This is implied by the inequality $|\textit{co-stack}_{\mathcal{U}}(S)| \leq 1 + 2$(the number of locally-maximal repeat regions in S). The term $|\mathcal{V}_{\mathcal{U}}|$ accounts for the constant contribution of 1 from each $S \in \mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$, and $2|\mathcal{E}_{\mathcal{U}}|$ corresponds to the right term because $\mathcal{E}_{\mathcal{U}}$ represents the total set of locally-maximal repeat regions in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. □

Proposition 3. *Given $\mathcal{G}_{\mathcal{U}}$, enumerating co-stack$_{\mathcal{U}}(S)$ for some $v_S \in \mathcal{V}_{\mathcal{U}}$ takes $O(|$outgoing edges of $v_S|)$ time.*

Proof. This is derived from the proof of $|\textit{co-stack}_{\mathcal{U}}(S)| \leq 1 + 2 \cdot$ the number of locally-maximal repeat regions in S in Appendix B.1. □

Since k-mers expressed by each stack of S inherit characteristics of S, such as frequency and extensions within $\mathcal{U}$, algorithms can leverage this information to compute *k-mer quantities*, by considering k-mers in the same stack as having similar roles in the k-mer-based objects. We further introduce four applications motivated by biological problems that benefit from this methodology.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

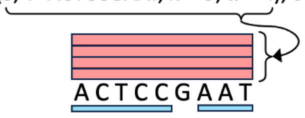
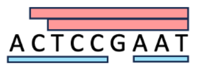
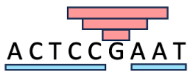
Types	*Co-occurrence stacks (red) of* S = ACTCCGAAT	k	Rate	Count	Expressed k-mers
Type 0	(S, T=ACTCCGAAT, k^*=9, d^*=4), δ_l=0, δ_r=0 	9	-1	1	ACTCCGAAT
		8		2	ACTCCGAA,CTCCGAAT
		7		3	ACTCCGA,CTCCGAA,TCCGAAT
		6		4	ACTCCG,CTCCGA,TCCGAA,CCGAAT
Type 1	(S, T=CTCCGAAT, k^*=5, d^*=2), δ_l=1, δ_r=0 	5	0	4	CTCCG,TCCGA,CCGAA,CGAAT
		4		4	TCCG,CCGA,CGAA,GAAT
Type 2	(S, T=CCGAA, k^*=3, d^*=3), δ_l=1, δ_r=1 	3	1	3	CCG, CGA, GAA
		2		2	CG,GA
		1		1	G

Figure 5: Three types of maximal co-occurrence stacks of S (sets of red rectangles), given two locally-maximal repeat regions in S (blue rectangles). The Rate column describes the change in the count of expressed k-mers as the k-mer size increases by 1. Hence, the rate is $1 - \delta_l - \delta_r$, and it controls the vertical shapes of stacks.

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

The number of distinct k-mers is commonly used to assess the complexity and structure of pangenomes and metagenomes [44, 13]. A recent study by [10] spans multiple k contexts, suggesting that the number of k-mers across all k-mer sizes should be used to estimate genome sizes. The following algorithm counts distinct k-mers for all k-mer sizes in time $O(|\mathcal{G}_\mathcal{U}|)$. We assume the maximum possible k-mer size, k_{max}, is always less than $|\mathcal{E}_\mathcal{U}|$, which is typical for most sequencing data and genome references.

The core idea leverages the change rate of k-mers within each co-occurrence stack, thereby eliminating the need to consider k-mer counts individually. The number of k-mers expressed by each type-0, 1, or 2 co-occurrence stack changes by -1, 0, or +1 as k increases by 1, respectively, as detailed in the Rate column of Figure 5. This property is utilized in lines 4-7, and hence the loop in line 2 runs in $O(|\mathcal{G}_\mathcal{U}|)$ time.

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

```
    Input: A Prokrustean 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 vectors of length $k_{max}$ populated with 0's: $C$, $\partial C$, and $\partial\partial C$.
2   for $v_S \in \mathcal{V}_\mathcal{U}$ do
3       for $((v_S, i, j), k^*, d^*) \in co\text{-}stack_\mathcal{U}(S)$ do
4           $\delta = \mathbf{1}_{i \neq 1} + \mathbf{1}_{j \neq size(v_S)} - 1$ // the change rate of the number of $k$-mers expressed by the stack
5           $C[k^*{-}d^*{+}1]{+}{=}(j{-}i{-}k^*{+}2){-}\delta \cdot (d^*{-}1)$ // the number of $k$-mers expressed by the stack for the lowest
              $k$-mer size
6           $\partial\partial C[k^* - d^* + 2]{+}{=}\delta$ // apply the change rate for the lowest k size
7           $\partial\partial C[k^* - 1]{-}{=}\delta$ // remove the change rate for the highest k size
8   for $k = 2 \ldots k_{max}$ do
9       $\partial C[k]{+}{=}\partial C[k-1]{+}\partial\partial C[k]$ // total change rate of number of $k$-mers at each $k$
10      $C[k]{+}{=}C[k-1]{+}\partial C[k]$ // total number of $k$-mers at each $k$
11  return $C$
```

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

The Bray-Curtis dissimilarity, defined with k-mers, is a frequency-based metric that measures the dissimilarity between biological samples. The values are sensitive to the choice of k-mer sizes, so they are often calculated with multiple k-mer sizes to check the influence [47] [23] [36], yet most state-of-the-art tools utilize a fixed k-mer size [7].

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

We computed the Bray-Curtis dissimilarity for all k-mer sizes of samples A and B using the Pro$k$rustean graph constructed from the union of their sequence sets, $\mathcal{U}_A$ and $\mathcal{U}_B$. This graph tracks the origins of sequences, computing the following quantities:

$$BC(\mathcal{U}_A, \mathcal{U}_B, k) = 1 - \frac{{}_{k\text{-mer in } \mathcal{U}_A\cup\mathcal{U}_B} 2 \cdot \min(\text{freq}_A(k\text{-mer}), \text{freq}_B(k\text{-mer}))}{{}_{k\text{-mer in } \mathcal{U}_A\cup\mathcal{U}_B} \text{freq}_A(k\text{-mer}) + \text{freq}_B(k\text{-mer})}$$

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

```
Input: A Prokrustean 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 $k_{max}$: $N$,$\partial N$,$\partial\partial N$,$D$,$\partial D$,$\partial\partial D$. # numerator, denominator
2  Compute frequencies $freq_A(v_S)$ and $freq_B(v_S)$ for all $v_S \in \mathcal{V}_\mathcal{U}$.
3  for $v_S \in \mathcal{V}_\mathcal{U}$ do
4      for $((v_S, i, j), k^*, d^*) \in co\text{-}stack_\mathcal{U}(S)$ do
5          # Imitate the value assignments to $C$, $\partial\partial C$ at lines 3-6 in Algorithm 2.
6          Assign to $N$, $\partial\partial N$ with all values weighted with $min(freq_A(v_S), freq_B(v_S))$
7          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      # Imitate the value assignments to $C$ using $\partial C$,$\partial\partial C$ at lines 8-9 in Algorithm 2.
10     Assign to $N$ using $\partial N$, $\partial\partial N$
11     Assign to $D$ using $\partial D$, $\partial\partial D$
12 return $N/D$
```

Recall that the value assignments in line 5 and 6 utilized co-occurrence stacks for counting k-mers in Algorithm 2. The values $min(freq_A(v_S), freq_B(v_S))$ and $freq_A(v_S) + freq_B(v_S)$ are assigned instead of a constant 1, because they grows linear to the number of co-occurring k-mers weighted by the frequency-based values. Consequently, vectors N and D contains values of the numerator and denominator part of the Bray-Curtis dissimilarities for all k-mer sizes.

Typically, biological experiments compute these values for pairs of multiple samples. The proposed algorithm can be extended to consider multiple samples, requiring $O(|\text{samples}|^2 \cdot |\mathcal{G}_\mathcal{U}|)$ time.

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 [20], represents a simple path that cannot be extended further, reflecting a topological characteristic of the graph. More maximal unitigs generally indicate more complex graph structures, which significantly impact genome assembly performance [14, 3]. Their number across multiple k-mer sizes reflects the influence of k-mer sizes on the complexity of assembly.

The idea is that maximal unitigs are implied by tips, convergences, and divergences, which are indicated by type-0 or 1 co-occurrence stacks. For instance, consider a string S in $\mathcal{U} \cup \mathcal{R}_\mathcal{U}$ with multiple right extensions, such as C and T. If a suffix region in S of length l is an locally-maximal repeat region, it suggests that a k-co-occurring suffix of S where $k > l$ corresponds to a vertex in the respective de Bruijn graph of order k with right extensions C and T. Therefore, identifying the locally-maximal repeat regions on suffixes and prefixes of strings in $\mathcal{U} \cup \mathcal{R}_\mathcal{U}$ is sufficient.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

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

```
Input: A Prokrustean graph $(\mathcal{V}_\mathcal{U}, \mathcal{E}_\mathcal{U})$, $k_{max}$
Output: Numbers of maximal unitigs of de Bruijn graphs of orders $k = 1, \dots, k_{max}$
1  Make 0 vectors of length $k_{max}$: $C$,$\partial C$.
2  Compute numbers of left/right extensions $left(v_S)$ and $right(v_S)$ for all $v_S \in \mathcal{V}_\mathcal{U}$.
3  for $v_S \in \mathcal{V}_\mathcal{U}$ do
4      for $((v_S, i, j), k^*, d^*) \in co\text{-}stack_\mathcal{U}(S)$ do
5          if $i = 1$ and $(left(v_S) = 0$ or $left(v_S) > 1)$ # tips or convergence then
6              $C[k^* - d^* + 1]$+=1
7              $\partial C[k^*]$−=1
8          if $j = size(v_S)$ and $right(v_S) > 1$ # divergence then
9              $C[k^* - d^* + 1]$+=$right(v_S)$
10             $\partial C[k^*]$−=$right(v_S)$
11 for $k = 2 \dots k_{max}$ do
12     $C[k]$+=$C[k-1]$+$\partial C[k]$
13 return $C$
```

4.6. Application: Computing vertex degrees of overlap graph

Overlap graphs are extensively utilized in genome and metagenome assembly, alongside de Bruijn graphs. Although their definition appears unrelated to k-mers, we can view them as representing suffix-prefix k-mers of sequences in $\mathcal{U}$. Overlap information is particularly crucial for metagenomics in read classification [16, 41, 31, 2] and contig binning [45, 32]. 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 $O(|\mathcal{U}|^2)$. A significant drawback of overlap graphs is their quadratic growth in the number of reads, $O(|\mathcal{U}|^2)$. In contrast, the Pro*k*rustean graph, which encompasses the overlap graph as a hierarchical subgraph, requires only $O(|\mathcal{G}_\mathcal{U}|)$ space. This structure enables the efficient computation of vertex degrees in the overlap graph.

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.

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

Starting from $v_S \in \mathcal{V}_\mathcal{U}$, explore a suffix path, meaning recursively choose edges that are locally-maximal repeat regions on suffices. 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 $O(|\mathcal{G}_\mathcal{U}|)$.

5. Experiments and Results

Here, we implement and test against various data sets the construction and four applications of the Pro*k*rustean graph. Datasets were randomly selected to represent a broad range of sizes, sequencing technologies, and biological origins. Both reference genomes and sequencing data were used to demonstrate the scalability of the Pro*k*rustean graph. 16 human pangenome and 3,500 metagenome references were used, and diverse sequencing datasets were collected, 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.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

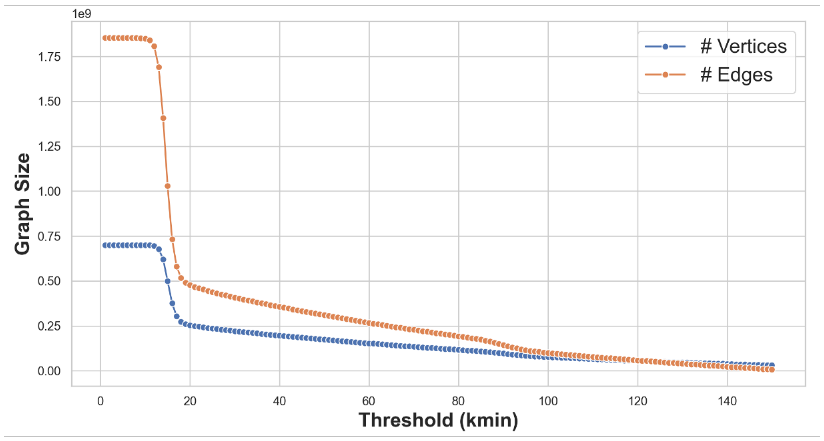


Figure 6: Sizes of Pro*k*rustean graphs (number of vertices and edges) versus k_{min} values for a short read dataset ERR3450203. The size of the graph drops around k_{min} = [11, ..., 19], meaning the maximal repeats and locally-maximal repeat regions are dense when k_{min} is under 11. The number of edges falls again around k_{min} = 100 because the graph eventually becomes fully disconnected.

5.1. *Prokrustean graph construction with k_{min}*

The construction algorithm is introduced later (Section 6). Recall that the Pro*k*rustean graph grows with the number of maximal repeats, i.e., $O(|\mathcal{G}_\mathcal{U}|) = O(|\Sigma| \cdot |\mathcal{R}_\mathcal{U}|)$. The size of $\mathcal{G}_\mathcal{U}$ can be controlled by dropping maximal repeats of length below k_{min}. This threshold limits the length of locally-maximal repeat regions to k_{min} or above, capturing a subgraph of the Pro*k*rustean graph, and hence achieves $O(|\mathcal{G}_\mathcal{U}|) = O(|\Sigma| \cdot |\mathcal{R}_\mathcal{U}(k_{min})|)$ space while computing *k-mer quantities* for $k = k_{min}, \ldots, k_{max}$. Figure 6 shows how the graph size of short sequencing reads of length 151 decreases as k_{min} increases.

Additionally, for 3500 E. coli genomes with $k_{min} = 10$, we observed that $|\mathcal{V}_\mathcal{U}| < 0.01N$ and $|\mathcal{E}_\mathcal{U}| < 0.03N$. Figure 7 shows how references of pangenome and metagenome grow. Again, the growth of the graph follows the growth of $|\mathcal{R}_\mathcal{U}|$.

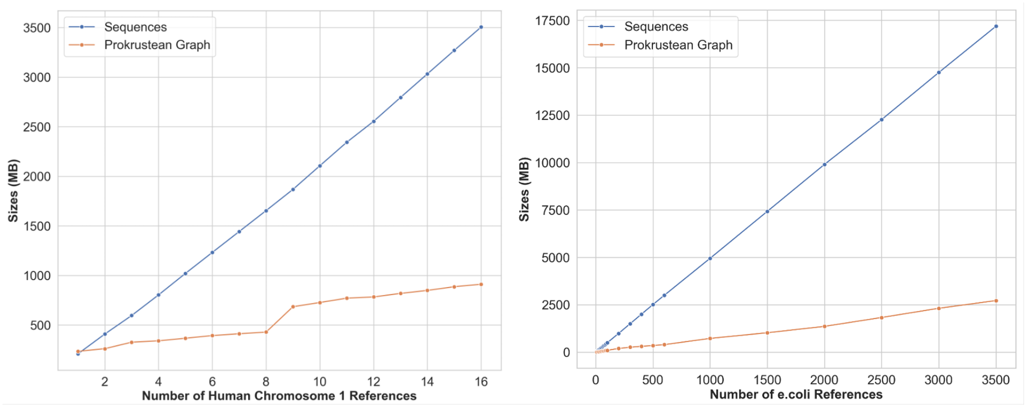


Figure 7: Sizes of Pro*k*rustean graphs as a function of sequences added. Left: Pro*k*rustean graphs constructed using an accumulation of 16 human chromosome 1 references. Right: same but generated using 3500 E. coli references exibiting an increased growth rate reflecting E. coli species diversity.

Table 1 shows that with some practical k_{min} values applied, the size of the Pro*k*rustean graphs stays around the size of their inputs. It is observed that $|\mathcal{G}_\mathcal{U}| \ll N$, with $|\mathcal{V}_\mathcal{U}| < 0.2N$ and $|\mathcal{E}_\mathcal{U}| < 0.5N$ with short reads.

Subsequent sections cover the results of four applications. Note that, to our knowledge, there are no existing computational techniques that perform the exact same tasks for a range of *k*-mer sizes, making performance com-

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

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 BWTs. Symbols M, B, mb, and gb mean millions, billions, megabytes, and gigabytes. N denotes the cumulative sequence length of $\mathcal{U}$ and we can see the output is comparable to N.

parisons 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 computational tools, such as the use of only ACGT (i.e. no "N") and canonical *k*-mers in KMC. Consequently, we tested correctness on GitHub using our brute-force implementations.

5.2. *Result: Counting distinct k-mers for $k = k_{min}, \dots, k_{max}$*

We employed KMC [28], 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	memory	threads
SRR20044276	1...150	5m	837mb	22	**0.36s**	80mb	8
ERR3450203	1...150	70m	12gb	22	**106s**	11gb	8
SRR18495451	30...50000	days	-	22	**88s**	24gb	8
SRR7130905	30...150	66m	12gb	22	**46s**	8.5gb	8
SRR21862404	30...150	112m	13gb	22	**74s**	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.

5.3. *Result: Computing Bray-Curtis dissimilarities for $k_{min}, \dots, k_{max}$*

Bray-Curtis dissimilarity is a popular metric used in metagenomics analysis [23, 36]. In each reference we found, dissimilarity scores were presented for a single *k*-mer size. Figure 8 shows that dissimilarities between four example metagenome samples are not consistent in that at some *k*-mer sizes, one observed pattern of dissimilarity is completely flipped at another *k*-mer size. I.e. 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, and the graph construction took around 1 hour. Computing the same quantity takes around 5 to 8 minutes for a fixed *k*-mer size with Simka [7], and no library computes it across *k*-mer sizes.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

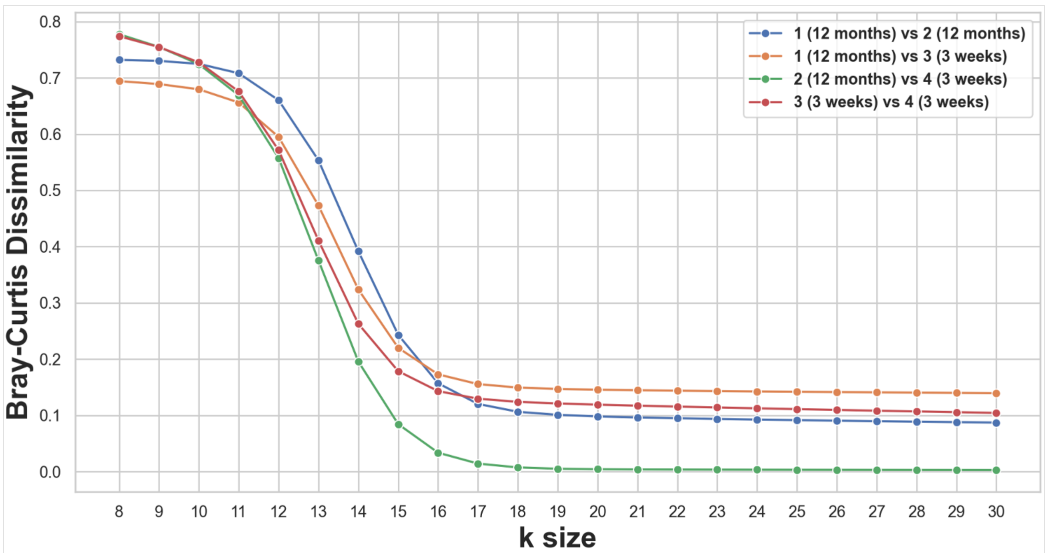


Figure 8: Bray-Curtis dissimilarities between four samples derived from infant fecal microbiota, which were studied in [36]. Samples include two from 12-month-old human subjects and two from 3-week-olds. The relative order between values shift as k changes, e.g. the dissimilarity between sample 2 and 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.

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

We utilized GGCAT [20], a mature library optimized for constructing compacted de Bruijn graphs at a fixed k-mer size. Since maximal unitigs are vertices of a compacted de Bruijn graph, we measured their de Bruijn graph construction time. Again, the Pro*k*rustean graph becomes more efficient as additional k-mer sizes are included in the computation. The following table displays the comparison, and Figure 9 illustrates some of the outputs.

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

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

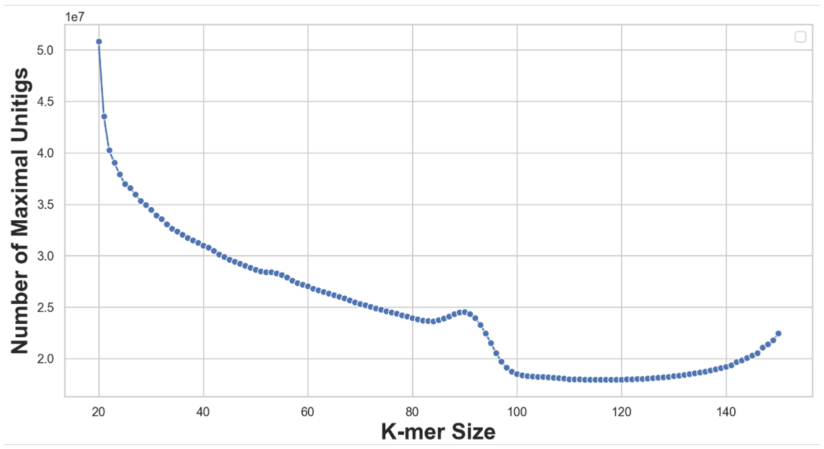


Figure 9: The number of maximal unitigs in de Bruijn graphs of order $k = 20, \dots, 150$ of metagenomic short reads (ERR3450203). A complex scenario is revealed: The decrease of the numbers fluctuates in $k = 30, \dots, 60$, and the peak around $k = 90$ corresponds to the sudden decrease in maximal repeats. Lastly, further disconnections increase contigs and eventually make the graph completely disconnected.

5.5. *Result: Counting vertex degrees of overlap graph of threshold k_{min}*

Here, we describe computing the vertex degrees of an overlap graph using a threshold of k_{min} on a metagenomic short read dataset. Limiting the task to vertex degree counting, $O(|\Sigma||\mathcal{R}_{\mathcal{U}}|)$ 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 10 used 8.5 gigabytes memory and 1 minute to count all vertex degrees.

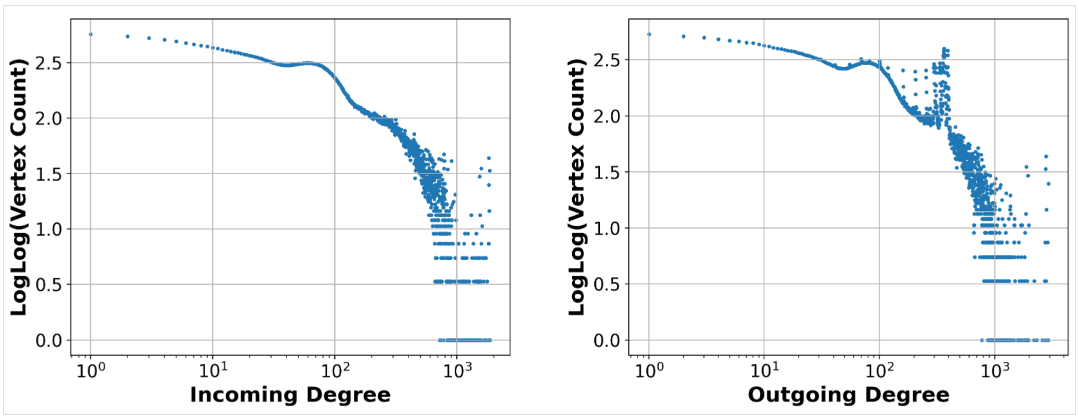


Figure 10: Vertex degrees of the overlap graph of metagenomic short reads (ERR3450203). The number of vertices per each degree is scaled as $\log \log y$. There is a clear peak around 10^2 at both incoming and outgoing degrees, which may imply dense existence of abundant species. The intermittent peaks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

6. Pro*k*rustean graph construction

Recall that Section 2.2 addressed the limitations of LCP-based substring indexes in computing $co\text{-}substr_{\mathcal{U}}(S)$, which motivated the adoption of the Pro*k*rustean graph. LCP-based representations inherently provide a "bottom-up" approach and are at best able to access incoming edges of v_S in the Pro*k*rustean graph. In contrast, the Pro*k*rustean graph supports a "top-down" approach by accessing $co\text{-}substr_{\mathcal{U}}(S)$ through co-occurrence stacks whose number depends on the outgoing edges of v_S.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

We start with a more straightforward, but less efficient approach: Section 6.1 employs affix trees—the union of the suffix tree of $\mathcal{U}$ and the suffix tree of its reversed strings—to extract the Pro*k*rustean graph. Although affix trees allow straightforward operations supporting bidirectional extensions above suffix trees, they are resource-intensive in practice. Section 6.2 refines the approach using the BWT of $\mathcal{U}$, incorporating an intermediate model to compensate the bidirectional context lost in moving away from affix trees. Section 6.3 then briefly introduce the implementation, covering recent advancements in BWT-based computations supporting it.

6.1. Extracting Prokrustean graph from two suffix trees

For ease of explanation, we use two suffix trees to represent the affix tree. Let $\mathcal{T}_{\mathcal{U}}$ denote the generalized suffix tree constructed from $\mathcal{U}$. The reversed string of a string S is denoted by $rev(S)$, and the reversed string set $rev(\mathcal{U})$ is $\{rev(S) \mid S \in \mathcal{U}\}$. A node in a tree is represented as $node(S)$ if the path from the root to the node spells out the string S, with the considered tree being always clear from the context. We assume affix links are constructed, which map between $node(S)$ in $\mathcal{T}_{\mathcal{U}}$ and $node(rev(S))$ in $\mathcal{T}_{rev(\mathcal{U})}$ if both nodes exist in their respective trees.

Additionally, a string preceded/followed by a letter represents an extension of the string with that letter, as in σS or $S\sigma$. Similarly, concatenation of two strings S and S' can be written as SS'. An asterisk (*) on a substring denotes any number (including zero) of letters extending the substring, as in $S*$.

6.1.1. Vertex set

Collecting maximal repeats of $\mathcal{U}$ requires information about $|occ_{\mathcal{U}}(\sigma R)|$ and $|occ_{\mathcal{U}}(R\sigma)|$ for each $\sigma \in \Sigma$, for any considered substring R. Most substring indexes built above LCPs provide access to occurrences through the LCP intervals, and the number of occurrences is represented by the length of the corresponding LCP intervals. The reverse representation $\mathcal{T}_{rev(\mathcal{U})}$ is not required yet, implying that the same information is accessible via single-directional indexes like the BWT in the next section.

6.1.2. Edge set

A more nuanced approach is required to extract the edge set of a Pro*k*rustean graph. Since an edge in a suffix tree basically implies co-occurrence, it sometimes directly indicates a locally-maximal repeat region, providing a subset of substrings that co-occur within their extensions. That is, an edge from $node(S)$ to $node(S')$ in $\mathcal{T}_{\mathcal{U}}$ implies that S may represent a locally-maximal repeat region in S' capturing a prefix S of S'. However, different scenarios may arise—sometimes the reverse direction should be considered, or both directions, or neither. These variations depend on the combinations of letter extensions around the substring. We present a theorem that delineates these cases, visually explained in Figure 11 for intuition.

We define terms distinguishing occurrence patterns of extensions. A string R is left-maximal if $|occ_{\mathcal{U}}(\sigma R)| < |occ_{\mathcal{U}}(R)|$ for every $\sigma \in \Sigma$, and left-non-maximal with σ if $|occ_{\mathcal{U}}(\sigma R)| = |occ_{\mathcal{U}}(R)|$, and right-maximality and right-non-maximality are symmetrically defined.

Theorem 6. *Consider a maximal repeat $R \in \mathcal{R}_{\mathcal{U}}$ and letters $\sigma_l, \sigma_r \in \Sigma$. The following three cases define a bijective correspondence between selected edges in the suffix trees ($\mathcal{T}_{\mathcal{U}}$ and $\mathcal{T}_{rev(\mathcal{U})}$) and the edges ($\mathcal{E}_{\mathcal{U}}$) of the Prokrustean graph of $\mathcal{U}$.*

Case 6.1 *(Prefix Condition) The following three statements are equivalent.*

- *(a) $R\sigma_r$ is left-maximal.*
- *(b) There exists a string T of form $R\sigma_r*$ in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ such that an edge from $node(R)$ to $node(T)$ is in $\mathcal{T}_{\mathcal{U}}$.*
- *(c) There exists an edge $e_{T,i,j}$ in $\mathcal{E}_{\mathcal{U}}$ satisfying $str(T, i, j+1) = R\sigma_r$ and $i = 1$.*

Case 6.2 *(Suffix Condition) The following three statements are equivalent.*

- *(a) $\sigma_l R$ is right-maximal.*
- *(b) There exists a string T of form $*\sigma_l R$ in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ such that an edge from $node(rev(R))$ to $node(rev(T))$ is in $\mathcal{T}_{rev(\mathcal{U})}$.*
- *(c) There exists an edge $e_{T,i,j} \in \mathcal{E}_{\mathcal{U}}$ satisfying $str(T, i-1, j) = \sigma_l R$ and $j = |T|$.*

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

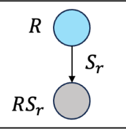
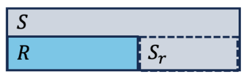
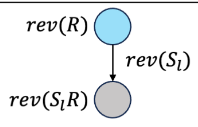
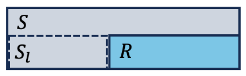
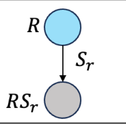
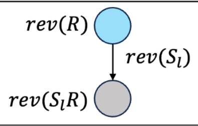
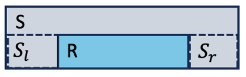
Scenarios	Suffix tree	Reversed suffix tree	Locally-maximal repeat regions
1. An edge in the suffix tree implies a locally-maximal repeat region at a prefix.			$(S, 1, \|R\|)$ where $S := RS_r$ 
2. An edge in the reversed suffix tree implies a locally-maximal repeat region at a suffix.			$(S, \|S_l\| + 1, \|S\|)$ where $S := S_lR$ 
3. One edge in the suffix tree and one in the reversed suffix tree imply a locally-maximal repeat region not extending to both ends.			$(S, \|S_l\|+1, \|S\| - \|S_r\|)$ where $S := S_lRS_r$ 

Figure 11: Deriving locally-maximal repeat regions from edges of suffix trees. From a node of a maximal repeat R or $rev(R)$, the extension S is identified by adding strings on edge annotations. Then, the three scenarios depict how each edge(s) locates each locally-maximal repeat region that captures R within S.

Case 6.3 *(Interior Condition) The following three statements are equivalent.*

(a) $R\sigma_r$ is left-non-maximal with σ_l and $\sigma_r R$ is right-non-maximal with σ_r.

(b) There exist strings S_l and S_r, and $T := S_lRS_r$ of form $*\sigma_lR\sigma_r*$ in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ such that an edge from node(R) to node(RS_r) is in $\mathcal{T}_{\mathcal{U}}$ and an edge from node$(rev(R))$ to node$(rev(S_lR))$ is in $\mathcal{T}_{rev(\mathcal{U})}$.

(c) There exists an edge $e_{T,i,j}$ in $\mathcal{E}_{\mathcal{U}}$ satisfying $str(T, i-1, j+1) = \sigma_lR\sigma_r$.

Proof. Refer to Appendix C.1 for the proof, which is straightforward but tedious. □

The computation of extracting the edge set of the Pro*k*rustean graph is realized by collecting incoming edges for each vertex: Explore $\mathcal{T}_{\mathcal{U}}$ to identify each maximal repeat $R \in \mathcal{R}_{\mathcal{U}}$, and verify the maximality conditions in items labeled (a) in the theorem. Upon satisfying (a) of any case, use the corresponding (b) to identify a superstring T from the suffix trees $\mathcal{T}_{\mathcal{U}}$ and $\mathcal{T}_{rev(\mathcal{U})}$. The corresponding (c) then confirms that the region of R within T is indeed a locally-maximal repeat region in $\mathcal{U}$. The precise region (T, i, j) is inferred from the decomposition of T outlined in (b). For instance, if T is S_lR as in Case 9.2, then (T, i, j) is the suffix region $(T, |S_l| + 1, |T|)$, so an edge from v_T to v_R is labeled $(|S_l| + 1, |T|)$.

This approach covers every edge in the Pro*k*rustean graph. Observe that items labeled (c) distinguish locally-maximal repeat regions by the letters on immediate left or right. Each region is unique with respect to the letter extension, as previously established in the proof of Theorem 4. Hence, for each maximal repeat R, the conditions met in (a) have bijectively mapped incoming edges to v_R, thus the entire edge set $\mathcal{E}_{\mathcal{U}}$ is identified through Theorem 9.

We have leveraged a bidirectional substring index, but if only one direction is supported, i.e., $\mathcal{T}_{rev(\mathcal{U})}$ is unavailable, finding T as outlined in (b) becomes challenging. This limitation with single-directional substring indexes motivates the development of an enhanced algorithm.

6.2. *Extracting Prokrustean graph from Burrows-Wheeler transform*

This section addresses two issues of the previous computation when implemented in practice. Firstly, suffix tree representations generally consume substantial memory, which hinders scalability when handling large genomic datasets such as pangenomes. Secondly, bidirectional indexes are more space-consuming, expensive to build, and less commonly implemented than single-directional indexes. Instead, BWTs enable traversal of the same suffix tree structure using sublinear space relative to the input sequences, and are supported by a range of actively improved construction algorithms.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Since the bidirectional extensions in items of (b) in Theorem 9 are not directly applicable to the BWT or other single-directional substring indexes, we utilize a subset of occurrences of maximal repeats as an intermediate representation of locally-maximal repeat regions. This approach necessitates a consistent way of utilizing suffix orders within the generalized suffix arrays of $\mathcal{U}$.

Before describing the construction of the Pro*k*rustean from a BWT, we need the following definitions.

Definition 9. *Let the rank of a region (S, i, j) be the rank of the suffix of S starting at i in the generalized suffix array of $\mathcal{U}$, which is implied by a substring index being used. Then, the first occurrence of a string R, $occ_{\mathcal{U}}(R)_{\mathbf{1}}$, is the lowest ranked occurrence among $occ_{\mathcal{U}}(R)$.*

Note that suffix ordering is consistent only within a sequence but varies across sequences depending on implementations. For example, among BWTs that employ different strategies, some consider the global lexicographical order, yielding $occ_{\mathcal{U}}(AA)_{\mathbf{1}} = (\text{CCAA}, 3, 4)$ for $\mathcal{U} = \{\text{GGAA}, \text{CCAA}\}$, while others impose a strict order on input sequences, such as $\mathcal{U} = \{\text{GGAA}\#_1, \text{CCAA}\#_2\}$, resulting in $occ_{\mathcal{U}}(AA)_{\mathbf{1}} = (\text{GGAA}, 3, 4)$. This variability is extensively discussed in [18]. In any scenario, the identical Pro*k*rustean graph will be generated as long as the first occurrences are consistently considered.

The following intermediate model achieves the goal by collecting specific occurrences of maximal repeats utilizing first occurrences. These occurrences are designed to form one-to-one correspondences with the edges of the Pro*k*rustean graph.

Definition 10. *$ProjOcc_{\mathcal{U}}$ is a subset of occurrences of strings in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$, where elements are specified as follows:*

1. *For every sequence $S \in \mathcal{U}$, the entire region of the sequence, i.e., $(S, 1, |S|) \in ProjOcc_{\mathcal{U}}$.*
2. *For every maximal repeat $R \in \mathcal{R}_{\mathcal{U}}$ and letters $\sigma_l, \sigma_r \in \Sigma$,*
   (1) If $R\sigma_r$ is left-maximal, then $(S, i, j-1) \in ProjOcc_{\mathcal{U}}$ given $(S, i, j) := occ_{\mathcal{U}}(R\sigma_r)_{\mathbf{1}}$.
   (2) If $\sigma_l R$ is right-maximal, then $(S, i+1, j) \in ProjOcc_{\mathcal{U}}$ given $(S, i, j) := occ_{\mathcal{U}}(\sigma_l R)_{\mathbf{1}}$.
   (3) If $R\sigma_r$ is left-non-maximal with σ_l and $\sigma_l R$ is right-non-maximal with σ_r, then $(S, i+1, j-1) \in ProjOcc_{\mathcal{U}}$ given $(S, i, j) := occ_{\mathcal{U}}(\sigma_l R \sigma_r)_{\mathbf{1}}$.

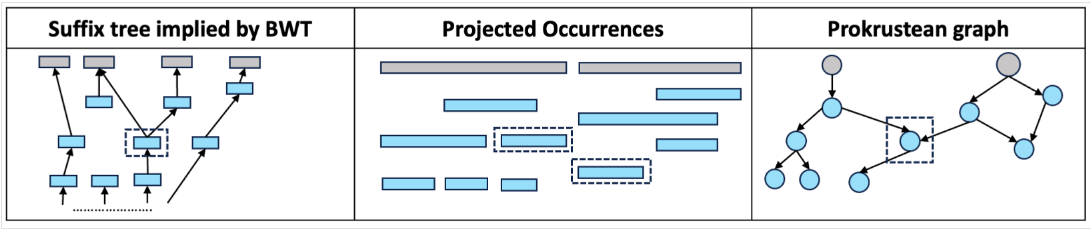


Figure 12: Construction of the Pro*k*rustean graph via projected occurrences. Gray vertices, depicted as either rectangles or circles, represent sequences in $\mathcal{U}$, while blue vertices denote maximal repeats of $\mathcal{U}$. The suffix tree is shown flipped to better illustrate the correspondence, providing intuition for the "bottom-up" approach. Dotted rectangles in each diagram refer to a maximal repeat R. First occurrences of substrings like $\sigma_l R$ and $R\sigma_r$ are identified through the construction of $ProjOcc_{\mathcal{U}}$. Then, the nested structure of projected occurrences derive incoming edges of v_R in the Pro*k*rustean graph.

Refer to Figure 12 for intuition that $ProjOcc_{\mathcal{U}}$ is indirectly deriving the locally-maximal repeat regions in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$. Although the same maximality conditions are used in both Theorem 9 and the definition of $ProjOcc_{\mathcal{U}}$, the occurrences in $ProjOcc_{\mathcal{U}}$ are organized in a nested manner, allowing locally-maximal repeat regions to be inferred through their inclusion relationships. So, $ProjOcc_{\mathcal{U}}$ is a projected image of the Pro*k*rustean graph of $\mathcal{U}$. The decoding rule for $ProjOcc_{\mathcal{U}}$, necessary for reconstructing the Pro*k*rustean graph, is articulated in the following proposition and theorem.

The following theorem elaborates the decoding rule. Let the relative occurrence of (S, i, j) within (S, i', j') be $(str(S, i', j'), i - i' + 1, j - i' + 1)$ defined only if $(S, i, j) \subsetneq (S, i', j')$.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Theorem 7. *$e_{T,p,q} \in \mathcal{E}_{\mathcal{U}}$, i.e., (T, p, q) is a locally-maximal repeat region in some $T \in \mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ if and only if there exist $(S, i, j), (S, i', j') \in ProjOcc_{\mathcal{U}}$ satisfying:*

- *$(S, i, j) \subsetneq (S, i', j')$, and*
- *the relative occurrence of (S, i, j) within (S, i', j') is (T, p, q), and*
- *no region $(S, x, y) \in ProjOcc_{\mathcal{U}}$ satisfies $(S, i, j) \subsetneq (S, x, y) \subsetneq (S, i', j')$.*

Proof. Refer to Appendix C.2 for the proof, which is straightforward but tedious. □

Therefore, two occurrences in their "closest" containment relationship in $ProjOcc_{\mathcal{U}}$ imply their strings form a locally-maximal repeat region. Since every occurrence in $ProjOcc_{\mathcal{U}}$ originates from either a maximal repeat or a sequence, each represents a vertex in the Pro*k*rustean graph. Therefore, collecting these relative occurrences constructs the edge set of the Pro*k*rustean graph. Refer to Appendix C for the algorithm.

6.3. Implementation

We briefly introduce the techniques implemented in our code base: https://github.com/KoslickiLab/prokrustean.

Firstly, the BWT is constructed from a set of sequences using any modern algorithm supporting multiple sequences [30, 17, 21]. The resulting BWT is then converted into a succinct string representation, such as a wavelet tree, to facilitate access to the "nodes" of the implied suffix tree. For this purpose, we used the implementation from the SDSL project [24]. The traversal is built on foundational works by Belazzougui et al. [4] and Beller et al. [6], as detailed by Nicola Prezza et al. [37]. Their node representation, based on LCP intervals, enables constant-time access to $occ_{\mathcal{U}}(R\sigma)$ and $occ_{\mathcal{U}}(\sigma R)$ for each substring R and letter $\sigma \in \Sigma$. This capability is crucial for verifying the maximality conditions outlined in Definition 10. Also, the first occurrence of so that the first occurrences $occ_{\mathcal{U}}(R\sigma)_{\mathbf{1}}$ and $occ_{\mathcal{U}}(\sigma R)_{\mathbf{1}}$ are identified as the start of the LCP interval of $R\sigma$ and σR, respectively, so $ProjOcc_{\mathcal{U}}$ can be built.

$ProjOcc_{\mathcal{U}}$ is collected by exploring the nodes of the suffix tree implicitly supported by the BWT of $\mathcal{U}$. The exploration requires $O(\log|\Sigma| \cdot |\mathcal{T}_{\mathcal{U}}|)$ time in total, because succinct string operations take $O(\log|\Sigma|)$ time. Identifying a maximal repeat takes $O(|\Sigma|)$ time and then collecting its occurrences in $ProjOcc_{\mathcal{U}}$ takes $O(|\Sigma|^2)$, which is obvious because each combination of occurrence, e.g., $|occ_{\mathcal{U}}(\sigma_l R\sigma_r)|$) is accessible in constant time. Therefore, constructing $ProjOcc_{\mathcal{U}}$ takes $O(\log|\Sigma| \cdot |\Sigma|^2 \cdot |\mathcal{T}_{\mathcal{U}}|)$ time and $O(|BWT| + |ProjOcc_{\mathcal{U}}|)$ space where $|BWT|$ is the size of the succinct string representing the BWT of $\mathcal{U}$.

Lastly, reconstructing the Pro*k*rustean graph using Theorem 10 takes $O(|ProjOcc_{\mathcal{U}}|)$ time, which is introduced in Appendix C. Note that the computation need not consider every combination of occurrences in $ProjOcc_{\mathcal{U}}$; it leverages the property that an occurrence $(S, i, j) \in ProjOcc_{\mathcal{U}}$ can imply locally-maximal repeat regions with up to two extensions in S found in $ProjOcc_{\mathcal{U}}$. Therefore, $ProjOcc_{\mathcal{U}}$ is grouped by each sequence $S \in \mathcal{U}$, and the edge set $\mathcal{E}_{\mathcal{U}}$ can be built from each group. Hence, the total space usage is $O(|BWT| + |ProjOcc_{\mathcal{U}}| + |\mathcal{G}_{\mathcal{U}}|)$, where $O(|ProjOcc_{\mathcal{U}}|) = O(|\mathcal{G}_{\mathcal{U}}|)$. The space usage is primarily output-dependent that around $2|\mathcal{G}_{\mathcal{U}}|$ in practice, and the most time-consuming part is the construction of $ProjOcc_{\mathcal{U}}$ via traversing the implicit suffix tree.

7. Discussion

We have introduced the problem of computing co-occurring substrings $co\text{-}substr_{\mathcal{U}}(S)$ (Section 2) and derived the Pro*k*rustean graph (Section 3). The graph facilitates computing $co\text{-}stack_{\mathcal{U}}(S)$ to efficiently express $co\text{-}substr_{\mathcal{U}}(S)$ (Theorem 8), which is used to implement algorithms computing k-mer-based quantities (Section 4.3, Section 4.4, Section 4.5, Section 4.6). Lastly, the graph was constructed from the BWT (Section 6).

It is worthwhile to note that the construction described in Section 6.1 highlights the limitations of LCP-based substring indices by comparing the differences between the Pro*k*rustean graph and suffix trees. Suffix trees cannot access the co-occurrence structure in constant time without the "top-down" representation supported by the Pro*k*rustean graph. Therefore, we conjecture that building multi-k methods with modern LCP-based substring indices is challenging regardless of the underlying approach.

The application Section 4.5 indicates that a Pro*k*rustean graph somehow access de Bruijn graphs of all orders. That is, a Pro*k*rustean graph has potential for advancing the so-called variable-order scheme. Indeed, a Pro*k*rustean of

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

$\mathcal{U}$ can be converted into the union of de Bruijn graphs of all orders. Since the Pro*k*rustean graph does not grow faster than maximal repeats, the new representation can help formulating multi-k methods to identify errors in sequencing reads, SNPs in population genomes, and assembly genomes.

Thus, the open problem 5 in [40] can be answered by the Pro*k*rustean graph, which calls for a practical representation of variable-order de Bruijn graphs that generates assembly comparable to that of overlap graphs. Recall that Section 4.6 analyzed the overlap graph of $\mathcal{U}$ from the Pro*k*rustean graph of $\mathcal{U}$. So, the two popular objects used in genome assembly are elegantly accessed through the Pro*k*rustean graph.

There is abundant literature analyzing the influence of k-mer sizes in many bioinformatics tasks. However, little effort has been made to derive rigorous formulations or quantities to explain these phenomena. It is a significant disadvantage to leave the effects of k-mer sizes shrouded in mystery, especially as many other biological assumptions are made in these tasks and complex pipelines. Understanding the usage of k-mers at least at the level of substring representations of the sequencing data or references is an essential initial step. Our framework is expected to contribute to forming this prerequisite so that further improvement involving biological knowledge is desired afterwards.

References

[1] 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.

[2] 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.

[3] Anton Bankevich, Andrey V Bzikadze, Mikhail Kolmogorov, Dmitry Antipov, and Pavel A Pevzner. Multiplex de bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature biotechnology*, 40(7):1075–1081, 2022.

[4] Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In *Proceedings of the forty-sixth Annual ACM Symposium on Theory of Computing*, pages 148–193, 2014.

[5] 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.

[6] Timo Beller, Simon Gog, Enno Ohlebusch, and Thomas Schnattinger. Computing the longest common prefix array based on the burrows–wheeler transform. *Journal of Discrete Algorithms*, 18:22–31, 2013.

[7] Gaëtan Benoit. Simka: fast kmer-based method for estimating the similarity between numerous metagenomic datasets. In *RCAM*, 2015.

[8] Maciej Besta, Raghavendra Kanakagiri, Harun Mustafa, Mikhail Karasikov, Gunnar Rätsch, Torsten Hoefler, and Edgar Solomonik. Communication-efficient jaccard similarity for high-performance distributed genome comparisons. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1122–1132. IEEE, 2020.

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

[10] Jessica K Bonnie, Omar Ahmed, and Ben Langmead. Dandd: efficient measurement of sequence growth and similarity. *bioRxiv*, pages 2023–02, 2023.

[11] Christina Boucher, Alex Bowe, Travis Gagie, Simon J Puglisi, and Kunihiko Sadakane. Variable-order de bruijn graphs. In *2015 data compression conference*, pages 383–392. IEEE, 2015.

[12] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de bruijn graphs. In *International workshop on algorithms in bioinformatics*, pages 225–235. Springer, 2012.

[13] 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.

[14] Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de bruijn graphs. *Genome biology*, 22:1–24, 2021.

[15] 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.

[16] Margherita Cavattoni and Matteo Comin. Classgraph: improving metagenomic read classification with overlap graphs. *Journal of Computational Biology*, 30(6):633–647, 2023.

[17] Davide Cenzato, Veronica Guerrini, Zsuzsanna Lipták, and Giovanna Rosone. Computing the optimal BWT of very large string collections. In *In Proc. of the 33rd Data Compression Conference, DCC 2023, 2023*, pages 71–80, 2023. `doi:10.1109/DCC55655.2023.00015`.

[18] Davide Cenzato and Zsuzsanna Lipták. A survey of bwt variants for string collections. *Bioinformatics*, page btae333, 2024.

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

[20] 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.

[21] Diego Díaz-Domínguez and Gonzalo Navarro. Efficient construction of the bwt for repetitive text using string compression. *Information and Computation*, 294:105088, 2023.

[22] 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.

[23] 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.

[24] 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.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

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

[26] Luiz Irber, Phillip T Brooks, Taylor Reiter, N Tessa Pierce-Ward, Mahmudur Rahman Hera, David Koslicki, and C Titus Brown. Lightweight compositional analysis of metagenomes with fracminhash and minimum metagenome covers. *bioRxiv*, pages 2022–01, 2022.

[27] 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.

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

[29] 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.

[30] Heng Li. Fast construction of fm-index for long sequence reads. *Bioinformatics*, 30(22):3274–3275, 2014.

[31] 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.

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

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

[34] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17:1–14, 2016.

[35] 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.

[36] 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.

[37] Nicola Prezza and Giovanna Rosone. Space-efficient computation of the lcp array from the burrows-wheeler transform. *arXiv preprint arXiv:1901.05226*, 2019.

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

[39] 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.

[40] 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.

[41] 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.

[42] Sebastian Schmidt, Shahbaz Khan, Jarno N Alanko, Giulio E Pibiri, and Alexandru I Tomescu. Matchtigs: minimum plain text representation of k-mer sets. *Genome Biology*, 24(1):136, 2023.

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

[44] 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.

[45] 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.

[46] 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.

[47] Hongxuan Zhai and Julia Fukuyama. A convenient correspondence between k-mer-based metagenomic distances and phylogenetically-informed β-diversity measures. *PLOS Computational Biology*, 19(1):e1010821, 2023.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

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 [36].

  ERS11976829, ERS11976830, ERS11976565, ERS11976566

B. Maximal co-occurrence stacks

B.1. The proof of co-stack$_{\mathcal{U}}(S)$ completely covering co-substr$_{\mathcal{U}}(S)$ and its bound.

Proposition 4. *A maximal k-co-occurring substring T and a maximal k'-co-occurring substring T' of S are identified by the same maximal co-occurrence stack if and only if they share identical boundary conditions: on both sides, they either extend to the end of S or intersect the same locally-maximal repeat region by $k-1$ and $k'-1$, respectively.*

Proof. Consider maximal k-co-occurring substring T and k'-co-occurring substring T' of S where $k < k'$. The forward direction is straightforward; a maximal co-occurrence stack (S, S', k^*, d^*), which identifies T and T, identifies S' as a k^*-co-occurring substring of S by definition, and S' either extends to an end of S or intersects an locally-maximal repeat region by k^*-1 on both sides according to Proposition 1. Then, as T and T' are identified as the (k^*-k+1)-th and $(k^*-k'+1)$-th elements by the stack, respectively, the rates δ_l and δ_r derived from the stack ensure T and T' sharing the equivalent side conditions with S' as described in Proposition 1.

For the reverse direction, consider three types of co-occurrence stacks: 1. $T = T' = S$, i.e., they extend to both ends of S. Then a type-0 stack identifies them. Whenever S k-co-occurs within S, the same holds for $k+1$ and above until $|S|$. Therefore, $(S, T := S, k^* := |S|, d^*)$ with some maximal d^* identifies T and T' .

2. T and T' are proper prefixes of S. Then a type-1 stack identifies them. The condition says there is a locally-maximal repeat region (S, i, j) intersecting the regions of T and T' in S by $k-1$ and $k'-1$, respectively. Without loss of generality, let the intersection be on the left of (S, i, j). Then, a k^*-co-occurring prefix of S with $k^* := |(S, i, j)|$ intersects (S, i, j) by k^*-1; if not, there must be a k^*-mer occurring more than $|occ_{\mathcal{U}}(S)|$ starting between positions 1 and $i-1$. But k-mers of T start in the positions too, so T does not k-co-occur within S, which leads to contradiction. Thus, $(S, T := str(S, 1, j-1), k^* := |(S, i, j)|, d^*)$ is a type-1 stack that identifies both T and T'.

3. T and T' are neither prefixes nor suffixes. Then a type-2 stack identifies them. There are two locally-maximal repeat regions (S, i, j) and (S, i', j') that intersect the regions of T and T' by $k-1$ and $k'-1$, respectively. Assuming (S, i, j) is smaller, a similar argument as in the previous paragraph shows that a k^*-co-occurring substring intersects both (S, i, j) and (S, i', j') by k^*-1 given $k^* := |(S, i, j)|$, so $(S, T := str(S, j-k^*+2, i'+k^*-2), k^* := |(S, i, j)|, d^*)$ identifies T and T'. □

This proposition implies that either zero, one, or two locally-maximal repeat regions in S characterize a co-occurrence stack by identifying substrings of identical side conditions. This corresponds to the classifications—type-0, type-1, and type-2. The proposition also implies that the number of maximal co-occurrence stacks of S does not grow faster than the number of locally-maximal repeat regions in S.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

Theorem 8. *$co\text{-}stack_{\mathcal{U}}(S)$ completely and disjointly cover co-occurring substrings of S, i.e.,*

$$co\text{-}substr_{\mathcal{U}}(S) = \bigsqcup_{(S,T,k^*,d^*)\in co\text{-}stack_{\mathcal{U}}(S)} \{\textit{substrings expressed by } (S,T,k^*,d^*)\}.$$

Also,

$$|co\text{-}stack_{\mathcal{U}}(S)| \leq 1 + 2 \cdot (\textit{the number of locally-maximal repeat regions in } S)$$

Proof. $co\text{-}stack_{\mathcal{U}}(S)$ completely and disjointly covers $co\text{-}substr_{\mathcal{U}}(S)$: any k-mer occurring in $\mathcal{U}$ co-occurs within an exactly one string S in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ by Theorem 1. The k-mer extends to a unique superstring that maximally k-co-occurs within S because appearing in two distinct maximally k-co-occurring substrings imply multi-occurrence of the k-mer in S. The superstring is identified by a unique maximal co-occurrence stack by Proposition 4. So, the k-mer is expressed by exactly one maximal co-occurrence stack.

To demonstrate the inequality, first, consider a type-0 stack on a string S, where all identified substrings are S itself by definition, i.e., extend to ends of S. A single type-0 stack identifies them all by Proposition 4. Hence, just one type-0 stack identifies them all by Proposition 4.

Next, for a type-1 or type-2 maximal co-occurrence stack (S, T, k^*, d^*), we show its longest identified substring always intersects some locally-maximal repeat region of length k^* by exactly $k^* - 1$. Since a proper type-1 stack identifies every k-co-occurring prefix (or suffix) of S whose region intersects the same locally-maximal repeat region by $k - 1$, by Proposition 4, the largest order k^* works too. Then, k^* should be the length of the locally-maximal repeat region; otherwise, we can find a $(k^* + 1)$-co-occurring prefix (or suffix) of S, denoted T', intersecting the locally-maximal repeat region by k^*, so $(S, T', k^* + 1, d^* + 1)$ is a proper co-occurrence stack, resulting in the original stack non-maximal. Similarly, a type-2 stack identifies substrings that intersect two locally-maximal repeat regions on both sides, so checking the length of the smaller region be k^* permits a similar deduction.

Note that each left and right side of a locally-maximal repeat region of length l intersect the region of at most one maximally l-co-occurring substring by exactly $l - 1$. Since such maximal intersection happens with every type-1 or type-2 maximal co-occurrence stack at least once, the number of such stacks is bounded by twice the number of locally-maximal repeat regions, thus is 2(the number of outgoing edges of v_S in $\mathcal{E}_{\mathcal{U}}$) in total.

Lastly, enumerating $co\text{-}stack_{\mathcal{U}}(S)$ is efficient. A type-0 stack is uniquely defined for a string S where the depth d^* is one more than the length of the largest locally-maximal repeat region. Each side of a locally-maximal repeat region maximally intersects the largest substring identified in at most one stack, either type-1 or type-2. Without loss of generality, a type-1 stack is defined on the left side of a locally-maximal repeat region if it is the largest one among those found on its left, while a type-2 stack pairs with the nearest locally-maximal repeat region of at least the same size on its left. These properties can be verified by a single scan of the locally-maximal repeat regions of a string. □

C. Pro*k*rustean graph construction

C.1. The proof of suffix trees to Prokrustean graph

Theorem 9. *Consider a maximal repeat $R \in \mathcal{R}_{\mathcal{U}}$ and letters $\sigma_l, \sigma_r \in \Sigma$. The following three cases define a bijective correspondence between selected edges in the suffix trees ($\mathcal{T}_{\mathcal{U}}$ and $\mathcal{T}_{rev(\mathcal{U})}$) and the edges ($\mathcal{E}_{\mathcal{U}}$) of the Prokrustean graph of $\mathcal{U}$.*

Case 9.1 *(Prefix Condition) The following three statements are equivalent.*

(a) $R\sigma_r$ is left-maximal.

(b) There exists a string T of form $R\sigma_r*$ in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ such that an edge from $node(R)$ to $node(T)$ is in $\mathcal{T}_{\mathcal{U}}$.

(c) There exists an edge $e_{T,i,j}$ in $\mathcal{E}_{\mathcal{U}}$ satisfying $str(T, i, j + 1) = R\sigma_r$ and $i = 1$.

Case 9.2 *(Suffix Condition) The following three statements are equivalent.*

(a) $\sigma_l R$ is right-maximal.

(b) There exists a string T of form $*\sigma_l R$ in $\mathcal{U} \cup \mathcal{R}_{\mathcal{U}}$ such that an edge from $node(rev(R))$ to $node(rev(T))$ is in $\mathcal{T}_{rev(\mathcal{U})}$.

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

(c) *There exists an edge $e_{T,i,j} \in \mathcal{E}_\mathcal{U}$ satisfying $str(T, i-1, j) = \sigma_l R$ and $j = |T|$.*

Case 9.3 *(Interior Condition) The following three statements are equivalent.*

(a) *$R\sigma_r$ is left-non-maximal with σ_l and $\sigma_r R$ is right-non-maximal with σ_r.*

(b) *There exist strings S_l and S_r, and $T := S_l R S_r$ of form $*\sigma_l R \sigma_r *$ in $\mathcal{U} \cup \mathcal{R}_\mathcal{U}$ such that an edge from $node(R)$ to $node(RS_r)$ is in $\mathcal{T}_\mathcal{U}$ and an edge from $node(rev(R))$ to $node(rev(S_l R))$ is in $\mathcal{T}_{rev(\mathcal{U})}$.*

(c) *There exists an edge $e_{T,i,j}$ in $\mathcal{E}_\mathcal{U}$ satisfying $str(T, i-1, j+1) = \sigma_l R \sigma_r$.*

Proof. Case 9.1 $(a) \Longrightarrow (b)$: We proceed arguing in the contrapositive. Assume $T := RS_r \notin \mathcal{U} \cup \mathcal{R}_\mathcal{U}$ which is non-maximal on either left or right. Observe that the definition of the construction of the suffix tree $\mathcal{T}_\mathcal{U}$ implies $|occ_\mathcal{U}(R\sigma_r)| = |occ_\mathcal{U}(RS_r)|$ and RS_r being right-maximal. Since RS_r is right-maximal, RS_r must be left-non-maximal with some $\sigma \in \Sigma$, so $|occ_\mathcal{U}(\sigma RS_r)| = |occ_\mathcal{U}(RS_r)|$. Since $|occ_\mathcal{U}(R\sigma_r)| = |occ_\mathcal{U}(RS_r)|$, $|occ_\mathcal{U}(\sigma RS_r)| = |occ_\mathcal{U}(RS_r)| = |occ_\mathcal{U}(R\sigma_r)|$ meaning $R\sigma_r$ is left-non-maximal with some σ.

Case 9.1 $(b) \Longrightarrow (c)$: Since $R, T \in \mathcal{U} \cup \mathcal{R}_\mathcal{U}$ holds and $T[|R|+1] = \sigma_r$, showing $(T, 1, |R|)$ is a locally-maximal repeat region is enough to show $e_{T,1,|R|}$ satisfies the statement. Let $T := RS_r$. First, the region's string R occurs more than RS_r because R is a maximal repeat. Next, as $(RS_r, 1, |R|)$ is a prefix region, every extension includes the smallest extension $(RS_r, 1, |R|+1)$, so the string of any of its extensions occurs at least $|occ_\mathcal{U}(R\sigma_r)|$, but $\mathcal{T}_\mathcal{U}$ implies $|occ_\mathcal{U}(R\sigma_r)| = |occ_\mathcal{U}(RS_r)|$, so the extensions co-occur within RS_r. Hence, $(T, 1, |R|)$ is a locally-maximal repeat region and $str(T, 1, |R|+1) = R\sigma_r$.

Case 9.1 $(c) \Longrightarrow (a)$: Since $T, 1, |R|$ is a locally-maximal repeat region, its extension captures $R\sigma_r$, so $|occ_\mathcal{U}(R\sigma_r)| = |occ_\mathcal{U}(T)|$ holds. Also, $T \in \mathcal{U} \cup \mathcal{R}_\mathcal{U}$ implies T is left- and right-maximal, so $|occ_\mathcal{U}(\sigma T)| < |occ_\mathcal{U}(T)|$ holds for any $\sigma \in \Sigma$. Therefore, $|occ_\mathcal{U}(\sigma R\sigma_r)| < |occ_\mathcal{U}(R\sigma_r)|$ for any $\sigma \in \Sigma$, hence $R\sigma_r$ is left-maximal.

**Case 9.2 is symmetrically argued in line with Case 9.1.

Case 9.3 $(a) \Longrightarrow (b)$: The non-maximalities in (a) imply $|occ_\mathcal{U}(\sigma_l R)| = |occ_\mathcal{U}(R\sigma_r)| = |occ_\mathcal{U}(\sigma_l R\sigma_r)|$. Also, by co-occurrences implied by two suffix trees, $|occ_\mathcal{U}(S_l R)| = |occ_\mathcal{U}(\sigma_l R)|$ and $|occ_\mathcal{U}(RS_r)| = |occ_\mathcal{U}(R\sigma_r)|$ hold. Hence, $|occ_\mathcal{U}(S_l R)| = |occ_\mathcal{U}(S_l RS_R)| = |occ_\mathcal{U}(RS_R)|$ is satisfied, meaning $S_l RS_R$ is left- and right-maximal that $T := S_l RS_R \in \mathcal{U} \cup \mathcal{R}_\mathcal{U}$.

Case 9.3 $(b) \Longrightarrow (c)$: The maximal repeat R occurs more than $S_l RS_r$ and the string of any extension occurs at least $|occ_\mathcal{U}(\sigma_l R)|$ or $|occ_\mathcal{U}(R\sigma_r)|$ times, while $|occ_\mathcal{U}(\sigma_l S)| = |occ_\mathcal{U}(S_l SS_R)| = |occ_\mathcal{U}(SS_R)|$ is implied by the two suffix trees. Hence, $(T, i, j) := (S_l RS_r, |S_l|+1, |S_l|+|R|)$ is a locally-maximal repeat region such that $str(T, i-1, j+1) = \sigma_l R\sigma_r$.

Case 9.3 $(c) \Longrightarrow (a)$: This statement is trivial because the locally-maximal repeat region implies $|occ_\mathcal{U}(\sigma_l R)| = |occ_\mathcal{U}(S_l RS_R)| = |occ_\mathcal{U}(R\sigma_r)|$, hence $|occ_\mathcal{U}(\sigma_l R)| = |occ_\mathcal{U}(R\sigma_r)| = |occ_\mathcal{U}(\sigma_l R\sigma_r)|$ holds, which implies non-maximalities on both sides of R. □

C.2. *The proof of projected occurrences to Prokrustean graph*

Below proposition is useful to prove the main theorem.

Proposition 5. *For every maximal repeat $R \in \mathcal{R}_\mathcal{U}$, $occ_\mathcal{U}(R)_1$ is in $ProjOcc_\mathcal{U}$.*

Proof. If a maximal repeat is a whole sequence in $\mathcal{U}$, it is trivially in $ProjOcc_\mathcal{U}$. So, assume every occurrence of R in $\mathcal{U}$ is extendable by some letters.

Let $(S, i, j) := occ_\mathcal{U}(R)_1$, and define $\sigma_l := S[i-1]$ and/or $\sigma_r := S[j+1]$. One of σ_l or σ_r might not be defined, so consider $(S, i-1, j) = occ_\mathcal{U}(\sigma_l R)_1$ without loss of generality.

If $\sigma_l R$ is right-maximal, then $occ_\mathcal{U}(R)_1$ trivially belongs to $ProjOcc_\mathcal{U}$ by 2-(2) of Definition 10. Otherwise, $\sigma_l R$ is right-non-maximal with some σ_r, so $(S, i, j+1) = occ_\mathcal{U}(R\sigma_r)_1$ must hold.

Consequently, whether $R\sigma_r$ is left-maximal or left-non-maximal with σ_l, $occ_\mathcal{U}(R)_1 \in ProjOcc_\mathcal{U}$ holds because conditions 2-(1) or 2-(3) of Definition 10 apply, respectively. The symmetric argument assuming $(S, i, j+1) = occ_\mathcal{U}(R\sigma_r)_1$ confirms the same results. □

Theorem 10. *$e_{T,p,q} \in \mathcal{E}_\mathcal{U}$, i.e., (T, p, q) is a locally-maximal repeat region in some $T \in \mathcal{U} \cup \mathcal{R}_\mathcal{U}$ if and only if there exist $(S, i, j), (S, i', j') \in ProjOcc_\mathcal{U}$ satisfying:*

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

- $(S, i, j) \subsetneq (S, i', j')$, *and*
- *the relative occurrence of* (S, i, j) *within* (S, i', j') *is* (T, p, q), *and*
- *no region* $(S, x, y) \in ProjOcc_{\mathcal{U}}$ *satisfies* $(S, i, j) \subsetneq (S, x, y) \subsetneq (S, i', j')$.

Proof. The forward direction is composed of three parts. We show there exists some $(S, i, j) \in ProjOcc_{\mathcal{U}}$ corresponding to (T, p, q) that is specified by Theorem 9, and then show there exists some $(S, i', j') \in ProjOcc_U$ such that the relative occurrence of (S, i, j) in the region is (T, p, q). Lastly, the third statement is shown.

First, there exists an occurrence $(S, i, j) \in ProjOcc_{\mathcal{U}}$ that captures the same string and shares some left or right extension with (T, p, q). (T, p, q) must be satisfying exactly one of three items labeled (c) in Case 9.1, Case 9.2, and Case 9.3 conditioned by extensions of (T, p, q). Then the corresponding item of (a) describes a maximality condition around R that is equally described in 2-(1), 2-(2), and 2-(3) in Definition 10 to derive elements in $ProjOcc_{\mathcal{U}}$. So, there is always some (S, i, j) in $ProjOcc_{\mathcal{U}}$ chosen for (T, p, q) by the agreement in letter extensions, i.e., either $S[i - 1] = T[p - 1]$ or $S[j + 1] = T[q + 1]$ or both holds.

Next, there exists some (S, i', j') such that the relative occurrence of (S, i, j) within (S, i', j') is (T, p, q). The construction of (S, i, j) showed either $S[i - 1] = T[p - 1]$ or $S[j + 1] = T[q + 1]$ holds, so assume $S[i - 1] = T[p - 1]$ without loss of generality. Since (T, p, q) is a locally-maximal repeat region, $str(T, p - 1, q)$ co-occurs within T and hence $str(S, i - 1, j)$ co-occurs within T. Therefore, (S, i, j) can be extended to some (S, i', j') so that T is captured and i is at $i' + p - 1$ reflecting the position of (T, p, q) within T. Similarly, $j = j' + q - 1$ holds, and hence (T, p, q) is recovered by $(str(S, i', j'), i - i' + 1, j - j' + 1)$.

We need $(S, i', j') \in ProjOcc_{\mathcal{U}}$ to finish showing the second statement. If T is in $\mathcal{U}$, then trivially $(S, i', j') = (S, 1, |S|)$ holds and $(S, 1, |S|) \in ProjOcc_{\mathcal{U}}$ is applied by item 1. in Definition 10. Otherwise, T is in $\mathcal{R}_{\mathcal{U}}$, then (S, i', j') is $occ_{\mathcal{U}}(T)_{\mathbf{1}}$ because the first occurrence of some subtring co-occurring within T was utilized for identifying (S, i, j). Proposition 5 confirms that $occ_{\mathcal{U}}(T)_{\mathbf{1}} \in ProjOcc_{\mathcal{U}}$.

Lastly, assume some (S, x, y) in $ProjOcc_{\mathcal{U}}$ satisfies $(S, i, j) \subsetneq (S, x, y) \subsetneq (S, i', j')$. Then a maximal repeat $R'(:= str(S, x, y))$ exists such that R' is a superstring of $R(:= str(T, p, q))$ and T is a superstring of R'. Then R' could be captured by extending (T, p, q) and R' occurs more than T, so (T, p, q) is not a locally-maximal repeat region, leading to a contradiction.

For the backward direction, assume (T, p, q) is not a locally-maximal repeat region for contradiction. Then some locally-maximal repeat region (T, p', q') must be found by extending (T, p, q) because $R := str(T, p, q)$ occurs more than T. Then by following the the same steps in the first paragraph utilizing Theorem 9 and Definition 10, (T, p', q') is used to identify an occurrence (S'', i'', j'') in $ProjOcc_{\mathcal{U}}$ that shares some letter extension. So, letting $R' := (T, p', q')$, a substring of form either $\sigma_l R'$ or $R' \sigma_r$ is used to find a first occurrence and the substring co-occurs within T, so (S'', x'', y'') appears within $occ_{\mathcal{U}}(T)_{\mathbf{1}}$ so that its relative occurrence within $occ_{\mathcal{U}}(T)_{\mathbf{1}}$ is (T, p', q'). Therefore, $(S'', i'', j'') \subsetneq occ_{\mathcal{U}}(T)_{\mathbf{1}}$. Also, $(S, i, j) \subsetneq (S'', i'', j'')$ because (T, p', q') is an extension of (T, p, q) that (S'', i'', j'') is an extension of (S, i, j) too. Lastly, $occ_{\mathcal{U}}(T)_{\mathbf{1}} = (S, i', j')$ indeed holds because of $(S, i, j) \subsetneq (S, i', j')$. That is, if $occ_{\mathcal{U}}(T)_{\mathbf{1}}$ is not (S, i', j') but some other occurrence of T in S'', then the corresponding occurrence of R in $ProjOcc_{\mathcal{U}}$ must be found in S'' instead of (S, i, j). Thus, the presence of (S'', i'', j'') leads to contradiction. □

bioRxiv preprint doi: https://doi.org/10.1101/2023.11.21.568151; this version posted December 20, 2024. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

C.3. The algorithm — projected occurrences to Prokrustean graph

The algorithm below reduces the task of reconstructing the Pro*k*rustean graph from $ProjOcc_{\mathcal{U}}$ to a problem designed for a specific string $S \in \mathcal{U}$. For the subset of regions $\{(S', i, j) \in ProjOcc_{\mathcal{U}} \mid S' = S\}$, determine their inclusion relationships to identify the closest related pairs. This is easily parallelized as it processes the intervals for each S separately. And the running time is linear to the size of the subset of regions per each S, hence $|ProjOcc_{\mathcal{U}}|$ in total.

An important assumption in the algorithm is that for any region $(S, i, j) \in ProjOcc_{\mathcal{U}}$, within the subset $\{(S', i, j) \in ProjOcc_{\mathcal{U}} \mid S' = S\}$, there are at most two regions (S, i', j') such that $(S, i, j) \subsetneq (S, i', j')$ and no (S, x, y) exists where $(S, i, j) \subsetneq (S, x, y) \subsetneq (S, i', j')$. It is straightforward to check the property.

Algorithm 5: Find recursive inclusions of intervals

```
1  Input: A set of intervals $I$ pre-ordered by first and then second element.
2  Output: Interval pairs such that each $(I, I')$ satisfies $I \subsetneq I'$ and no other interval $I''$ satisfies $I \subsetneq I'' \subsetneq I'$. [1]
3  Set $Ints := Ints[pos]$ = intervals in $I$ starting from $pos$. (accessed by $Ints[pos][idx]$).
4  Set $IdxMap := IdxMap[pos] = 0$ for each starting position $pos$ in $Ints$.
5  Set $PostMap := PostMap[pos] = nextPos$ per each adjacent positions ($pos$,$nextPos$) in $Ints$, and $nextPos$ is
     NULL if $pos$ is rightmost.
6  Set $curPos$ = The rightmost position of $Ints$.
7  Set $curIdx = 0$
8  Set $Pairs = \varnothing$
9  while $curIdx \neq NULL$ do
10     if $curIdx > 0$ then
11         // Adjacent intervals sharing start positions completely satisfy the conditions.
            $Pairs = Pairs \cup \{(Ints[curPos][curIdx-1], Ints[curPos][curIdx])\}$
12     Set $postPos = PostMap[curPos]$
13     while $postPos \neq NULL$ do
14         $postIdx = IdxMap[postPos]$ if $Ints[postPos][postIdx] \not\subset Ints[curPos][curIdx]$ then
15             break while
16         while $Ints[postPos][postIdx+1] \subset Ints[curPos][curIdx]$ do
17             $postIdx$ += 1
18         $Pairs = Pairs \cup \{(Ints[postPos][postIdx], Ints[curPos][curIdx])\}$
19         if $postIdx$ is NOT last in $Ints[postPos]$ then
20             $IdxMap[postPos] = postIdx + 1$
21         else
22             $IdxMap[postPos]$ =NULL
23             $PostMap[curPos] = PostMap[postPos]$
24         $postPos = PostMap[curPos]$
25     if $curIdx$ is NOT last in $Ints[curPos]$ then
26         $curIdx$ += 1
27     else if $curPos$ is not first in $Ints$ then
28         $curPos$= previous position of $curPos$ in $Ints$
29         $curIdx$= $IdxMap[curPos]$
30     else
31         $curIdx$= NULL
32 Return $Pairs$
```