Pangenome Barcoding using Strongly Unique K-mers

Shreeharsha G Bhat BE21B037[1], Pravar Vijaywargiya BS21B025[2]

1 Department of Biotechnology, Biological Engineering, Indian Institute of Technology, Madras, 600036, Chennai, India
2 Department of Biotechnology, Biological Sciences, Indian Institute of Technology, Madras, 600036, Chennai, India

Abstract

A central problem of pangenome analysis includes the identification and differentiation of various strains present in the pangenome. Strongly unique k-mers (SUKs) are short DNA sequences that are unique in the pangenome and also lack any close (Hamming distance-1) neighbors in the entire pangenome. Detecting SUKs that serve as barcodes for genomic strain identification can minimize the mapping ambiguity. Current algorithms that search for Strongly Unique Kmers work well on single genomes but do not take into consideration the high sequence similarities. In this study, we address the computational problem of efficiently identifying Strongly Unique Kmers in highly similar pangenomes. We develop 3 algorithms that utilize the similarities between genomes to iteratively check for hamming-distance-1 neighbours and minimize computations of previously encountered k-mers. Algorithm 1 is a brute-force approach, iteratively going through every genome. Algorithm 2 utilizes the tool PanKmer to create a presence/absence matrix and uses a top-down approach to identify unique k-mers and then check for the HammDist-1 neighbours. Algorithm 3 combines the results of the tool PanKmer and the Quarter Algorithm for finding Strongly Unique Kmers in each genome. Experiments were performed on bacterial pangenome datasets, with genome size minimized for computational convenience. All three algorithms generated identical SUK counts, but Algorithm 3 exhibited the best runtime. The runtimes decreased with increasing similarity in pangenome datasets. Different regions in the genome

Introduction

Pangenomes are a set of various genomes of a species. [1]. Pangenome analysis using k-mer decomposition methods is rapidly gaining popularity due to their decreased time and space complexities in comparision to graph based methods. K-mer decomposition of pangenomes has several applications in genome assembly [2], error correction [3], genome editing [4], taxonomy classification [5], and phylogeny reconstruction [6].

Unique k-mers are k-mers that appear once in a DNA sequence. Identifying Unique k-mers is one such k-mer analysis method that helps identify disease-related mutations [7]. Unique k-mers have also been used in graph-based bacterial strain identifications by classifying isolates based on the distinct kmers present in them [8]. They help in locating DNA fragments without alignment and ambiguity. However, they lack robustness since a single base pair change may change a unique k-mer to a k-mer already present in the genome. This may lead to incorrect read mapping based on k-mer composition. Strongly unique k-mers(SUKs) are unique k-mers with no Hamming-distance-1 neighbour. They are considered more robust since no neighbour with conflicting k-mer information exists for such k-mers. Identifying strongly unique kmers will also help create highly specific markers to identify genomic strains accurately [9]. They can act as strong primer binding locations for hybridization assays [10].

This uniqueness property of k-mers can be identified by counting k-mers. Several such tools and efficient methods exist for k-mer counting, such as Gerbil [11], KMC 2 and 3 [12,13], Jellyfish [14], hackgap [15]. But, it is sufficient to count up to 2 for strongly unique k-mer identification. The problem lies in finding the strongly unique k-mers with minimal comparisons of k-mers.

A brute force method for SUK identification involves checking the existence of every $3k$ neighbour for all k-mers present in the genome. Hence, the time complexity would be $\mathcal{O}(3k.N)$, where N is the length of the genome.

Faster methods to identify SUKs take a k-mer set as input and output the set of strongly unique k-mers in linear time, but with a smaller constant factor [16]. The two existing algorithms for swift SUK identification are the *Fourway* and the *Quarter* method. For a given input set of k-mers, both algorithms initialize by creating a lexicographically sorted list of k-mers and their reverse complements, which doubles the input size (worst case). The *Fourway* method runs a recursive call on the lexicographically ordered set with a pointer to the beginning of a collection of k-mers subset starting with A, C, T, and G to perform a 4-way comparison, similar to a multi-way merge sort. The *Quarter* algorithm performs pairwise comparisons on the last quarter and second-last quarter of the sequences in order to check for mismatches. These algorithms work better for single genomes, but eukaryotic pangenomes have been known to be highly similar and have close to 99% sequence similarity across the genomes [17].

PanKmer is a k-mer decomposition-based tool that creates a k-mer index for all k-mers present in the input set of genomes [18]. The output consists of a binary matrix where each row represents the presence/absence of a particular k-mer in all the genomes given in the input. This can help us easily identify unique k-mers. Finding SUKs would require further analysis.

Here, we develop three algorithms that can swiftly identify SUKs in highly similar pangenomes. The first algorithm analyses each genome separately in a brute-force fashion, while the other 2 algorithms use a presence/absence matrix to enumerate the Unique Kmers. To check for Strong Uniqueness, we use the brute force neighbour-checking algorithm and the quarter algorithm. The algorithm's time complexities were evaluated on biological datasets.

Materials and methods

The datasets used were of microbial pangenomes, *Mycoplasmoides genitalium* (580 kbp), *Brucella melitensis* (3.3 Mbp), and *Saccharomyces cerevisiae* (12 Mbp) from NCBI. While the dataset was too large to run all the algorithms, we performed experiments on reduced genome lengths (20 kbp) for each of these datasets.

Algorithm 1

The intuition behind the first algorithm is that, since most genomes are highly similar, most k-mers will repeat across genomes. Checking for the strong uniqueness property for these k-mers in multiple iterations will be redundant. The original Strongly Unique Kmers algorithms [16] operate on a single Kmer list. Running this algorithm on one genome at a time does not check the strong unique of kmers across genomes while running it on the full set of kmers in the pangenome will not provide us with barcoding kmers for each genome. We start by iterating through the k-mer list of each genome.

Once the SUKs in the first genome have been identified and the rest are marked as weak k-mers. When iterating through subsequent genomes, if a k-mer has already been encountered before, we directly mark it as weak. If not, this k-mer is a suitable candidate for strong uniqueness. At every iteration, we get a new set of possible candidates on which we run the brute force algorithm to identify SUKs.

Algorithm 1: Genome-wise Search

```
Input  : Genomes $G_1, G_2, ..., G_n$, k-mer length $k$
Output : Final set of strongly unique k-mers (SUK)
Initialize: // Process first genome
$(SUK, Weak) \leftarrow$ SUKAlgorithm($G_1$)
foreach genome $G_i \in \{G_2, ..., G_n\}$ do
    $NewlyIdentified \leftarrow \Phi$
    foreach k-mer $X \in$ ExtractKmers($G_i, k$) do
        if $X \in SUK$ or $X \in Weak$ then
            $Weak \leftarrow Weak \cup \{X\}$
            if $X \in SUK$ then
                $SUK \leftarrow SUK \setminus \{X\}$
        else
            $NewlyIdentified \leftarrow NewlyIdentified \cup \{X\}$
    $CandidatePool \leftarrow SUK \cup NewlyIdentified$
    $(UpdatedSUK, NewWeak) \leftarrow$ SUKAlgorithm($CandidatePool$)
    $SUK \leftarrow UpdatedSUK$
    $Weak \leftarrow Weak \cup NewWeak$
return $SUK$

SUKAlgorithm(sequence):
// Returns a tuple of (SUK_Set, Weak_Set)
ExtractKmers(genome, k):
// Generates all k-length subsequences
```

Algorithm 2

The approach for the second algorithm is more top-down. We start by running PanKmer on all the input genomes. Using the PanKmer output, which is a presence/absence matrix, we identify all the unique k-mers first. These are rows in the binary Matrix that contain a single 1-bit, indicating that the k-mer is present only in a single Genome. Finally, we search for the presence of Hamming-distance-1 neighbours in the full kmer array generated in the presence/absence matrix for each of the unique k-mers. This algorithm utilizes the Pankmer matrix to minimize the number of kmers for which Strong Uniqueness must be checked. The algorithm workflow for both the above algorithms is shown in Fig. 1. Initial testing for correctness is done using a **pangenome generator**. A Python function generates a random genome and multiple copies of that genome and randomly mutates m number of bases in the whole pangenome.

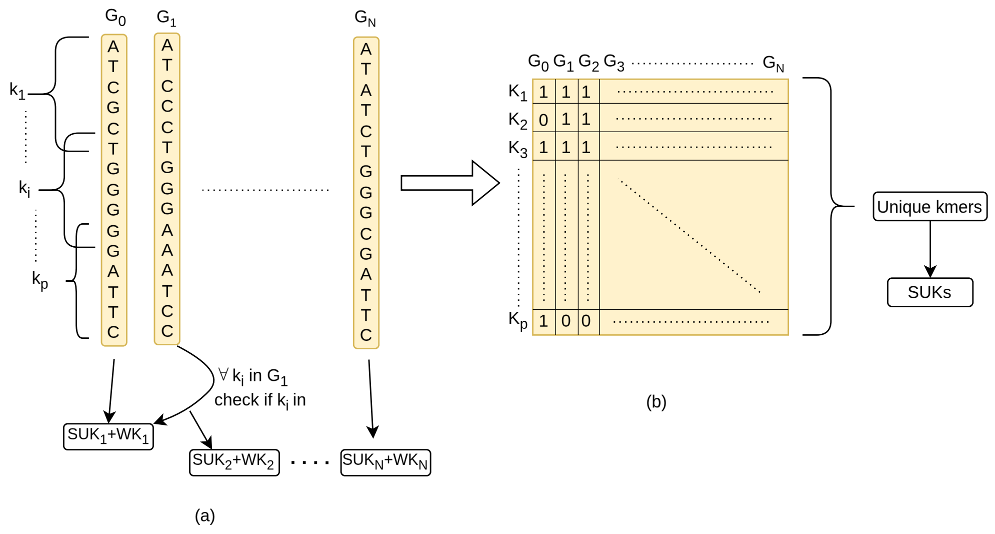

Fig 1. Figure depicting the algorithm workflow for **(a)** Genome-wide Search and **(b)** PanKmer Based algorithm

```
Algorithm 2: PanKmer based
Input  : Genome collection $G = \{G_1, G_2, ..., G_n\}$, k-mer length $k$
Output : Set of strongly unique k-mers (SUK)
Step 1: Presence/Absence Matrix Construction
$M \leftarrow$ RunPanKmer($G, k$) ;                // Get $m \times n$ binary matrix
Step 2: Unique K-mer Identification
$Candidates \leftarrow \Phi$
foreach row $r \in M$ do
    if $\sum_{j=1}^{n} r[j] = 1$ then
        $Candidates \leftarrow Candidates \cup \{$GetKmer($r$)$\}$
Step 3: SUK Identification
$SUK \leftarrow \Phi$
foreach k-mer $x \in Candidates$ do
    $isUnique \leftarrow$ True
    $Neighbors \leftarrow$ GenerateHammingNeighbors($x$)
    foreach $y \in Neighbors$ do
        if $y \in$ GetAllKmers($M$) then
            $isUnique \leftarrow$ False
            break
    if $isUnique$ then
        $SUK \leftarrow SUK \cup \{x\}$
return $SUK$
RunPanKmer(G, k):
// Generates $m \times n$ binary matrix where $M_{ij} = 1$ iff k-mer $i$ exists
   in genome $j$.  Handles reverse complements and canonical forms
GenerateHammingNeighbors($x$):
// Returns all k-mers with Hamming distance 1 from $x$
GetAllKmers(M):
// Returns set of all k-mers represented in matrix $M$
```

We also implemented a **3rd algorithm** using the Quarter method on the entire k-mer list for all genomes and their reverse complements. The intuition behind the quarter algorithm is that in the lexicographically ordered set of k-mers, if any two canonical k-mers k_1, k_2 have a Hamming distance of 1, then at least one of the pairs from $\{(k_1, rc(k_1)) \times (k_2, rc(k_2))\}$ has their SNP in the 3rd quarter or 4th quarter of the genome sequence. Here $rc(k)$ is the reverse complement of the k-mer k. If the difference is in the 4th quarter, we can bucket k-mers based on identical 1st three quarters and check pairwise comparisons in that particular bucket. If the difference is in the 3rd quarter, the pairwise comparisons must be done in the bucket containing k-mers that have the 1st,2nd, and 4th quarter matching. We can do this by swapping the sequence for the 3rd and 4th quarter and locally resorting with blocks that share the common 1st and 2nd quarter. This algorithm is able to combine the results of both tools for our requirement without significant changes to the PanKmer tool and the Quarter algorithm for swiftly identfiying strongly unique kmers.

For all three mentioned algorithms, the output not only includes a set of strongly unique k-mers, but also an index mapping it to the particular genome it belongs to. In this way, we are able to create a barcode for a specific pangenome.

While Pankmer has a Python module whose inputs are a set of genomes, the SUK algorithms (Fourway, Quarter) take a set of k-mers as inputs and output SUKs. Since we require a common method to compare runtime and accuracy, the implementation of all three algorithms was done in Python. The `NumPy` and `collections` libraries were used for data handling, while the `Bio` and `Pankmer` modules were used to parse genome files and to run PanKmer. The plots and time complexity analysis were done using the `matplotlib` and `Time` libraries, respectively.

Results & Discussion

The three algorithms were run on five genomes of the Mycoplasmoides genitalium dataset with k = 31 and a cutoff on genome size at 20 kbs, and on 5 genomes, Tab. 1. The splice taken to find SUKs was [30000 : 50000] in all genomes.

Strain ID	Algorithm 1	Algorithm 2	Algorithm 3
$L43967.2$	21	21	21
$CP003770.1$	76	76	76
$CP003771.1$	150	150	150
$CP003772.1$	14	14	14
$CP003773.1$	185	185	185
Time Taken	2825 sec	120 sec	20 sec

Table 1. Algorithmwise SUK counts and runtime for *Mycoplasmoides genitalium* dataset cutoff at 20 kbs genome size, $k = 31$

The results in Tab. 1 and Tab. 2 verify the correctness of all 3 algorithms. The same number of Strongly Unique Kmers have been obtained for each strain by all the algorithms. Runtime performance for all 3 algorithms shows that algorithm 3 works fastest. For the Brucella Melitensis datasets, the genome splices in various regions showed 100% identity for a few strains and 100% uniqueness for the remaining strains. (These results are not shown)

Since the dataset size in both Tab. 1 and Tab. 2 is the same, we can also infer the role of similarity in increasing the algorithm runtimes. The number of SUKs found is correlated to the runtime required.

Strain ID	Algorithm 2	Algorithm 3
CP026301.1	5589	5589
CP025097.1	159	159
CP089100.1	260	260
CP096554.1	13013	13013
LR813585.2	5417	5417
Time Taken	4265 sec	120 sec

Table 2. Algorithmwise SUK counts and runtime for Yeast dataset cutoff at 20 kbs genome size, k = 31

This result is observed more clearly in Tab. 3 where the dataset sizes are the same, but the average number of SUKs found is directly related to the algorithm runtime.

Tab. 3 also shows a glimpse of the similarity found in the genomes of a pangenome. The number of SUKs can also indicate variable or accessory regions of the genome. It is clear that the $[80k : 100k]$ splice contains a part of the Accessory Genome in the Mycoplasm Genitalium genome strains, and identification of these regions across the genome can lead to ease in downstream analysis on smaller regions as opposed to using graph-based methods for aligning all genomes in a pangenome.

Strain ID	0-20kbp	20-40kbp	40-60kbp	60-80kbp	80-100kbp
CP026301.1	0	76	22	0	982
CP025097.1	0	0	22	3	740
CP089100.1	86	50	349	84	1719
CP096554.1	25	14	0	0	1176
LR813585.2	61	207	0	31	1494
Time Taken	6.1 sec	12 sec	11.7 sec	6.3 sec	29.3 sec

Table 3. SUK counts for different regions in the *Mycoplasm Genitalium* dataset, cutoff at intervals of 20 kbps genome, $k = 31$

We also analysed the variation in the number of strongly unique k-mers with increasing k-mer size on the same biological dataset (Fig. 2). The number of SUKs found increases with increasing k-mer size, which can serve as a parameter to generate more specific barcodes for the same dataset. This would help in deciding appropriate k-values for different pangenomes based on similarity and genome size , to have the most efficient barcode with minimal k-mer length.

Runtime comparisons of all three algorithms were done on a self-created pangenome dataset with an increasing number of mutations, hence decreasing similarities in the pangenome (Fig. 3). As observed earlier, increasing uniqueness led to an increase in runtime for all 3 algorithms.

To observe the variation in runtime for larger dataset sizes, we ran the 3 algorithms on varying sizes (Fig. 4). The trend with increasing size is an increase in runtime. The similarity percentage was kept constant as size increased, to observe clear trends. While the runtime for Algorithm 3 was linear for these genome sizes, Algorithm 1 shot up with an exponential runtime in these datasets.

Further runtime analysis shows us that Algorithm 3, which uses the SUK identification method described in [16], runs faster than Algorithms 1 and 2. The runtime is linear for all three cases, but the constant factor is the lowest for Algorithm 3. One present limitation in our algorithms is that they create barcodes based on the available data. In case, a new genome is added to the dataset, the full algorithm needs

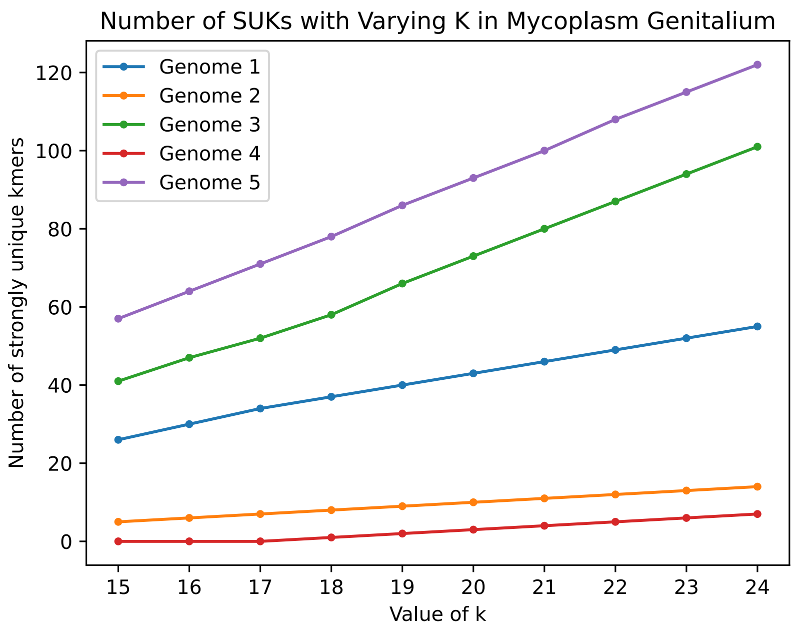

Fig 2. Figure shows the number of SUKs identified for increasing kmer lengths in Mycoplasmoides genitalium

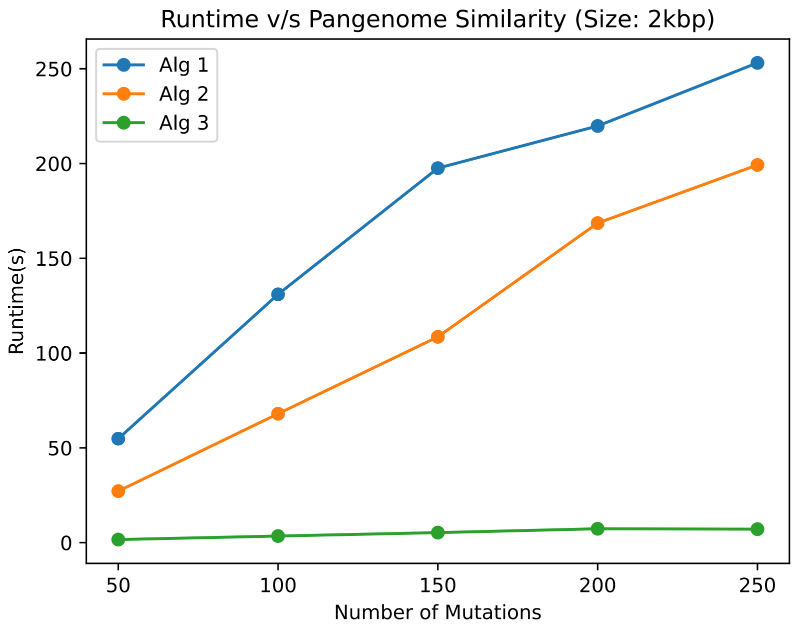

Fig 3. Runtime analysis for the three algorithms with decreasing similarity across genomes.

to be run again to generate specific barcodes.

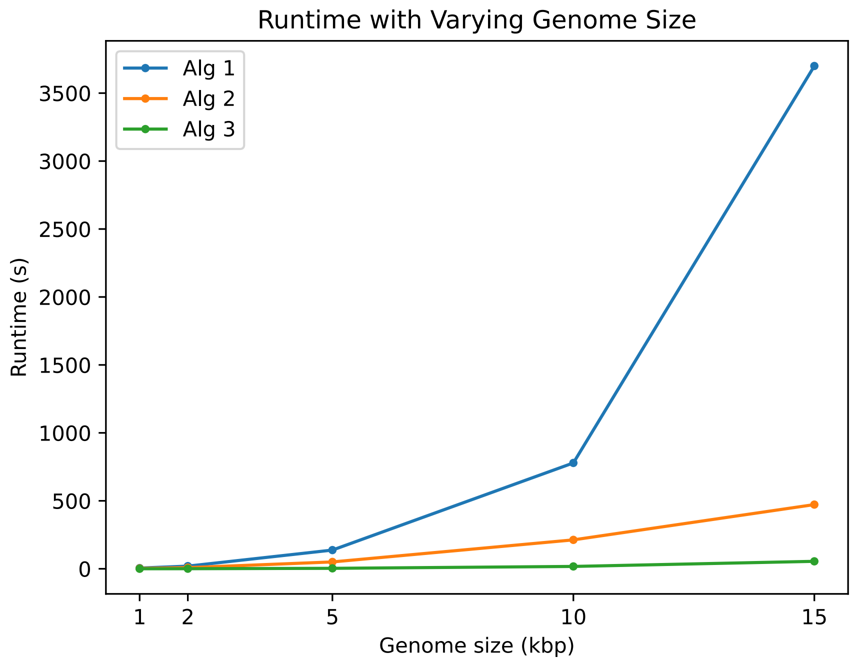

Fig 4. Runtime of the algorithms for varying size of a dataset with constant similarity

Conclusion

The algorithms developed here provide an easy method for strain identification and also to evaluate accessory regions in the genome without the need for full sequence alignment. The advantage of pangenomes being highly similar was observed as increasing similarity reduced the computations required to identify unique and strongly unique kmers. Varying the choice of k led to an increase in strongly unique kmers identified for the same dataset. Running the algorithm on splices of full genomes can help identify core and variable regions in the pangenome and can be a future direction, in order to complement further downstream analysis of these pangenomes.

Acknowledgments

We would like to thank Prof.Manikandan Narayanan for his invaluable guidance. Providing us with valuable insights and constructive feedback during our presentations helped us enhance our work. We would also like to thank the course teaching assistants for the same.

Author Contribution

- Shreeharsha G Bhat BE21B037: Algorithm Ideation and Debugging, Report writing, Literature Review.
- Pravar Vijaywargiya BS21B025: Dataset collection, Algorithm Implementations, Data Analysis, and interpretation.

GitHub Link

GitHub repository for all the algorithms and generated barcodes for the datasets: https://github.com/pravarv/Pangenome_Barcoding

Progress After Presentation

After the presentation, bugs in the implementation of Algorithms 1 and 2 were fixed. Algorithm 3 was developed and implemented. All 3 algorithms were tested on multiple self-generated datasets as well as biological datasets, for correctness. All algorithms were updated to output barcode k-mers to uniquely identify the pangenome strains.

Similar Work in Other Courses

This project has not been submitted for credit in any other course. All the work presented here is original and was undertaken specifically for this course.

References

1. Abondio P, Cilli E, Luiselli D. Human pangenomics: promises and challenges of a distributed genomic reference. Life. 2023;13(6):1360.
2. Ebler J, Ebert P, Clarke WE, Rausch T, Audano PA, Houwaart T, et al. Pangenome-based genome inference allows efficient and accurate genotyping across a wide spectrum of variant classes. Nature genetics. 2022;54(4):518–525.
3. Laehnemann D, Borkhardt A, McHardy AC. Denoising DNA deep sequencing data—high-throughput sequencing errors and their correction. Briefings in bioinformatics. 2016;17(1):154–179.
4. Papathanos PA, Windbichler N. Redkmer: an assembly-free pipeline for the identification of abundant and specific X-chromosome target sequences for X-shredding by CRISPR endonucleases. The CRISPR journal. 2018;1(1):88–98.
5. Simon HY, Siddle KJ, Park DJ, Sabeti PC. Benchmarking metagenomics tools for taxonomic classification. Cell. 2019;178(4):779–794.
6. Morgenstern B, Zhu B, Horwege S, Leimeister CA. Estimating evolutionary distances between genomic sequences from spaced-word matches. Algorithms for Molecular Biology. 2015;10:1–12.
7. Lee H, Shuaibi A, Bell JM, Pavlichin DS, Ji HP. Unique k-mer sequences for validating cancer-related substitution, insertion and deletion mutations. NAR cancer. 2020;2(4):zcaa034.
8. Roosaare M, Vaher M, Kaplinski L, Möls M, Andreson R, Lepamets M, et al. StrainSeeker: fast identification of bacterial strains from raw sequencing reads using user-provided guide trees. PeerJ. 2017;5:e3353.
9. Panyukov VV, Kiselev SS, Ozoline ON. Unique k-mers as strain-specific barcodes for phylogenetic analysis and natural microbiome profiling. International Journal of Molecular Sciences. 2020;21(3):944.
10. Martínez-Porchas M, Vargas-Albores F. An efficient strategy using k-mers to analyse 16S rRNA sequences. Heliyon. 2017;3(7).

11. Erbert M, Rechner S, Müller-Hannemann M. Gerbil: a fast and memory-efficient k-mer counter with GPU-support. Algorithms for Molecular Biology. 2017;12:1–12.

12. Deorowicz S, Kokot M, Grabowski S, Debudaj-Grabysz A. KMC 2: fast and resource-frugal k-mer counting. Bioinformatics. 2015;31(10):1569–1576.

13. Kokot M, Długosz M, Deorowicz S. KMC 3: counting and manipulating k-mer statistics. Bioinformatics. 2017;33(17):2759–2761.

14. Marcais G, Kingsford C. Jellyfish: A fast k-mer counter. Tutorialis e Manuais. 2012;1(1-8):1038.

15. Zentgraf J, Rahmann S. Fast gapped k-mer counting with subdivided multi-way bucketed cuckoo hash tables. In: 22nd International Workshop on Algorithms in Bioinformatics (WABI 2022). Schloss Dagstuhl–Leibniz-Zentrum für Informatik; 2022. p. 12–1.

16. Zentgraf J, Rahmann S. Swiftly Identifying Strongly Unique k-Mers. In: 24th International Workshop on Algorithms in Bioinformatics (WABI 2024). Schloss Dagstuhl–Leibniz-Zentrum für Informatik; 2024. p. 15–1.

17. Liao WW, Asri M, Ebler J, Doerr D, Haukness M, Hickey G, et al. A draft human pangenome reference. Nature. 2023;617(7960):312–324.

18. Aylward AJ, Petrus S, Mamerto A, Hartwick NT, Michael TP. PanKmer: k-mer-based and reference-free pangenome analysis. Bioinformatics. 2023;39(10):btad621.