CS249_A1 : k-mer index with minimizer

Omar Ahmed
MS in BioEngineering
omar.ahmed@kaust.edu.sa
KAUST ID: 212769

I. Objective

This assignment aims to implement and evaluate various approaches for metagenomic sequence classification using string matching and k-mer indexing techniques. The primary objective is to develop a system capable of accurately identifying the source organisms of DNA sequence reads from mixed samples. Through this implementation, we explore the efficiency and accuracy trade-offs of different classification methods, including exact string matching, k-mer indexing, and minimizer-based approaches, while comparing our implementations with industry-standard tools like BLAST and Kraken2. The code implementation is available here [1]

II. Introduction

Metagenomics involves the study of genetic material recovered directly from environmental samples, presenting unique computational challenges due to the presence of DNA from multiple organisms. In this project, we focus on the classification of short DNA sequence reads (typically 100-150bp) generated through shotgun sequencing. The core challenge is to efficiently determine the organism of origin for each read using reference genomes of known species.

We work with five well-characterized bacterial genomes representing different taxonomic groups and genomic properties: *E. coli K-12 MG1655*, *B. subtilis 168*, *P. aeruginosa PAO1*, *S. aureus NCTC 8325*, and *M. tuberculosis H37Rv*. Our dataset consists of two paired-end samples with 10,000 reads each: one without sequencing errors and one generated with an Illumina MiSeq error model.

The report covers three main approaches to read classification:

1) String matching-based classification using Uses the Burrows-Wheeler Transform (BWT) for efficient substring searching
2) K-mer index-based classification with exact matches
3) K-mer index with minimizers to improve memory efficiency

Finally, we compare our implementations with established bioinformatics tools and apply these methods to real-world metagenomic samples.

III. Metagenome Classification by String Matching

A. Multiple Matches and Exact Matching

For classification of metagenomic reads, I implemented two complementary approaches: (1) an FM-index for rapid string searching and (2) a resource-efficient exact matching algorithm using streaming and parallel processing.

1) Strategy for Handling Multiple Matches: When classifying reads from metagenomic samples, a key challenge is handling reads that match multiple reference genomes. This occurs due to evolutionary conservation of certain genomic regions across species. My implementation addresses this using a comprehensive tracking system:

1) Each read is compared against all reference genomes independently
2) Matches are stored in a dictionary mapping read indices to lists of matching organisms
3) Classifications are categorized as:
   - Unique matches: Reads matching exactly one organism
   - Multiple matches: Reads matching more than one organism
   - Unclassified: Reads with no matches to any reference genome

For biological accuracy, retaining the full list of matching organisms is crucial as it reflects genuine sequence homology rather than forcing an arbitrary assignment to a single species. This approach preserves valuable information about evolutionary relationships between the organisms in the sample.

2) Implementation of Exact Matching: My implementation uses two complementary string matching algorithms:

a) FM-Index Based Approach: I implemented a memory-efficient FM-index using the Burrows-Wheeler Transform (BWT) for fast substring searching:

- Uses chunking with a default size of 10,000,000 characters to handle large genomes
- Implements a sampled suffix array with configurable sampling rate (default 128) to reduce memory footprint
- Builds efficient checkpoint tables (C and Occ) for the backward search algorithm
- Processes text in manageable chunks to avoid memory overflow

The FM-index enables rapid backward search which is particularly efficient for exact matching operations, with time complexity O(m) where m is the pattern length, regardless of the text size.

b) Resource-Efficient Matching: For handling large datasets, I implemented a parallel processing approach:

- Uses concurrent processing with adaptive worker allocation based on available system memory

- Implements streaming for both genome and read files to minimize memory usage
- Processes reads in small batches (1,000 reads per batch) for better memory management
- Uses a simple yet effective exact matching algorithm that checks if a read is a substring of the genome chunk

3) Classification Results: Table I presents the classification results using exact string matching for both error-free and error-containing reads.

TABLE I: Classification Results for Exact String Matching

Classification	**Error-free**		**Error-containing**	
	Reads	**%**	**Reads**	**%**
E. coli K-12 MG1655	3,053	30.53	216	2.16
S. aureus NCTC 8325	512	5.12	37	0.37
M. tuberculosis H37Rv	506	5.06	42	0.42
B. subtilis 168	505	5.05	48	0.48
P. aeruginosa PAO1	503	5.03	35	0.35
Reads with multiple matches	9	0.09	1	0.01
Unclassified reads	4,930	49.30	9,623	96.23
Total	**10,000**	**100.00**	**10,000**	**100.00**

The results demonstrate a stark contrast between error-free and error-containing reads. With error-free reads, approximately 50.7% were successfully classified, with E. coli representing the majority (30.53%). However, with error-containing reads, the classification rate dropped dramatically to just 3.78%, highlighting a fundamental limitation of exact matching approaches when dealing with sequencing errors. This significant drop indicates that for real-world metagenomic data, which invariably contains sequencing errors, exact matching alone is insufficient.

The extremely low rate of multiple matches (0.09% in error-free reads) suggests that the 31bp k-mer length provided sufficient specificity to differentiate between the reference genomes in most cases. The relatively even distribution of matches across the non-E. coli organisms (approximately 5% each for error-free reads) indicates balanced representation in the simulated dataset.

The implementation of a memory-efficient FM-index enabled both matching and classification to be completed in a cumulative time of 560 seconds, with a peak memory usage of just 72.06 MB. The process was carried out in two phases: matching error-free and error-containing reads against the reference genomes. Most of the runtime was spent on constructing the FM-index, whereas the classification step itself took less than a second.

B. Comparison with BLAST

To validate and benchmark our implementation, I compared the performance and accuracy of our string matching algorithms with BLAST (Basic Local Alignment Search Tool), a widely used and highly optimized tool for sequence alignment.

1) BLAST Implementation: I implemented a BLASTN-based classification pipeline with parameters chosen to facilitate a fair comparison with our exact string matching approach:

- **Performance optimizations:** Used multithreading (4 threads) and efficient database indexing
- **Stringent matching criteria:** Set $-$ perc_identity as 100 to require 100% identity, similar to our exact matching
- **Forward strand only:** Used $-$strand plus to align with our implementation
- **Best match reporting:** Set $-$max_target_seqs as 1 to report only the best match for each read

2) Comparison of Results: Table II presents the classification results using BLAST for both error-free and error-containing reads.

TABLE II: Classification Results Using BLAST

Classification	**Error-free**		**Error-containing**	
	Reads	**%**	**Reads**	**%**
E. coli K-12 MG1655	3,071	30.71	333	3.33
S. aureus NCTC 8325	505	5.05	55	0.55
M. tuberculosis H37Rv	504	5.04	53	0.53
B. subtilis 168	507	5.07	60	0.60
P. aeruginosa PAO1	503	5.03	50	0.50
Reads with multiple matches	0	0.00	0	0.00
Unclassified reads	4,910	49.10	9,449	94.49
Total	**10,000**	**100.00**	**10,000**	**100.00**

Comparing Tables I and II reveals interesting similarities and differences:

1) **Error-free reads:** BLAST achieved slightly higher classification rates (50.9% vs. 50.7%) with a similar distribution across organisms.
2) **Error-containing reads:** BLAST significantly outperformed our approach (5.51% vs. 3.78%). This suggests that BLAST's seed-and-extend algorithm can better handle sequencing errors through its local alignment approach.
3) **Multiple matches:** BLAST reported no multiple matches, whereas our implementation identified a small percentage. This difference arises from BLAST's strategy of reporting only the highest-scoring alignment per read.

3) Performance Comparison: Performance metrics reveal key differences between the implementations:

- **Execution time:** BLAST completed the entire analysis in 34.83 seconds, which is significantly faster than our implementation that took several minutes to process the same dataset.
- **Memory usage:** BLAST exhibited excellent memory efficiency with a peak usage of only 62.47 MB which is better than my implementation due to less optimized data structures.
- **Scalability:** BLAST's memory usage decreased over time as shown in Figure 1, demonstrating efficient resource management for processing large datasets.

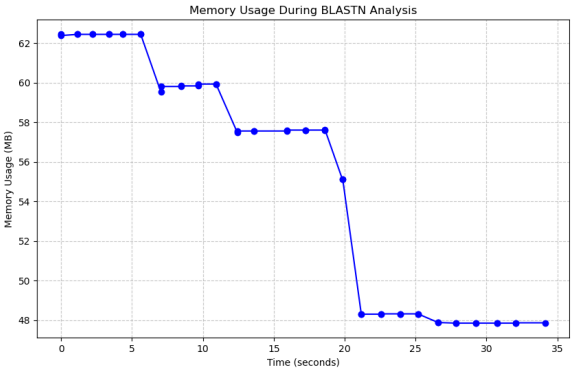

Fig. 1: The graph depicts memory usage during BLASTN analysis, showing peak consumption of approximately 62 MB during the initial database creation phase followed by step-wise decreases to a steady 48 MB. This pattern occurs because database creation requires building memory-intensive search indices upfront, while the subsequent read classification phase operates more efficiently with optimized algorithms that maintain only essential data structures in memory.

4) Analysis of Differences: The observed differences can be attributed to several factors:

1) **Algorithmic approach:** BLAST uses a seed-and-extend approach with optimized word matching, whereas our implementation uses a more straightforward string matching algorithm.
2) **Heuristic optimizations:** BLAST employs sophisticated heuristics and statistical models developed over three decades of refinement.
3) **Memory management:** BLAST uses disk-based indexing and efficient memory structures for database representation.
4) **Local alignment:** Despite requiring 100% identity, BLAST can identify high-quality partial matches, allowing it to classify more error-containing reads.

This comparison highlights the strengths of specialized bioinformatics tools like BLAST, while also validating our implementation's overall approach. Similar classification patterns for error-free reads confirm the accuracy of our algorithm, while performance differences emphasize the need for algorithmic optimizations to process large genomic datasets.

IV. Metagenomic classification by k-mer index

A. Building the k-mer Index

To improve classification efficiency, I implemented a k-mer index-based approach that precomputes a mapping between k-mers and their occurrences across reference genomes.

1) Data Structure Design: For the k-mer index, I used a hash-based data structure implemented as a Python dictionary with the following properties:

- **Keys:** Unique k-mers extracted from all reference genomes
- **Values:** Arrays of length 5 (one position per organism) storing the occurrence count of each k-mer
- **Default values:** A defaultdict implementation with zero counts [0,0,0,0,0] for unseen k-mers

This structure provides O(1) average-case lookup time for k-mers during classification while tracking organism-specific occurrence counts, which is critical for distinguishing between shared genomic regions.

2) Index Construction: The index construction process follows these steps:

- Extract all overlapping k-mers (k=31) from each genome by sliding a window of size k
- Filter out k-mers containing ambiguous bases (N's)
- For each k-mer, increment the corresponding organism's counter in the index
- Calculate statistics about k-mer distribution and sharing across genomes

3) Index Statistics: The k-mer index with k=31 produced the following statistics:

- Total unique k-mers across all genomes: 22,104,664
- Theoretical maximum possible k-mers: 4^{31} (approximately 4.6×10^{18})
- Memory usage: 1,364.47 MB
- Processing time: 44.14 seconds
- Serialized index size: 1,012.05 MB

The enormous discrepancy between the theoretical maximum (4^{31}) and the actual number of k-mers (22 million) can be attributed to two main factors:

- **Genome size constraints:** The combined length of all five genomes is approximately 22 million bp, setting a practical upper bound on the number of unique k-mers.
- **Biological constraints:** DNA sequences are not random; they contain functional regions, repetitive elements, and are subject to evolutionary pressures, resulting in non-uniform k-mer distributions.

4) K-mer Sharing Analysis: Analysis of k-mer sharing revealed extremely high specificity:

- 99.98% of k-mers appear in exactly one genome
- 0.02% appear in two genomes
- <0.01% appear in three or more genomes

This high specificity indicates that k=31 provides excellent discriminatory power for these genomes, with minimal overlap that could cause classification ambiguity.

B. Classification

I implemented two classification approaches using the k-mer index: a basic method based on majority voting and an advanced method with weighted voting.

1) Basic Classification Algorithm: The basic classification approach follows these steps:

- Extract all k-mers from each read
- For each k-mer, look up which organisms contain it
- Count matches to each organism
- Assign the read to the organism with the highest match count
- Mark reads with no matches as unclassified

2) Advanced Classification Algorithm: To handle the subtle case of shared k-mers more effectively, I implemented an advanced classification algorithm:

- Calculate k-mer specificity weights inversely proportional to the number of genomes containing each k-mer
- For each read, accumulate weighted votes for each organism based on matching k-mers
- Identify best matches (within 90% of maximum vote)
- Classify as ambiguous if multiple organisms have similar vote counts

3) Classification Results: Table III summarizes the classification results using the k-mer index approach. The advanced

TABLE III: Classification Results for K-mer Index

Classification	**Error-free(%)**	**Error-containing(%)**
E. coli	30.45	2.15
B. subtilis	5.05	0.48
P. aeruginosa	5.03	0.35
S. aureus	5.03	0.36
M. tuberculosis	5.06	0.42
Reads with multiple matches	0.09	0.01
Unclassified reads	49.29	96.23

classification method identified 10 ambiguous reads in the error-free dataset, representing cases where multiple organisms had similar k-mer matches due to conserved genomic regions. The k-mer indexing classification achieved virtually identical results to the string matching approach (Table I), confirming the consistency between the two methods. Both approaches show the same pattern of high classification rates for error-free reads (50.7%) and poor performance on error-containing reads (3.8%). This demonstrates that exact k-mer matching suffers from the same sensitivity to sequencing errors as exact string matching.

4) Performance Analysis: The k-mer index approach demonstrated significant performance advantages:

- Classification time: 0.73 seconds for 10,000 reads
- Memory usage during classification: 76.27 MB

These metrics represent a dramatic improvement over the string matching approach, highlighting the efficiency of precomputed indexes for metagenomic classification.

C. Minimizers

While the k-mer index provides excellent performance, its memory requirements remain substantial. To address this, I implemented a minimizer scheme that significantly reduces memory usage while maintaining classification accuracy.

1) Minimizer Implementation: A minimizer is a representative k-mer selected from a window of w consecutive k-mers based on a predefined ordering. My implementation uses the following parameters:

- k-mer length (k): 31
- Window size (w): 10
- Ordering: Lexicographic (smallest k-mer in each window)

For each window of w consecutive k-mers covering a substring of length $w + k - 1$, the implementation:

1) Extracts all valid k-mers (those without ambiguous bases) within the window
2) Identifies the lexicographically smallest k-mer as the minimizer
3) Adds this minimizer to the index, avoiding duplicates

This approach ensures that similar sequences share minimizers while drastically reducing the index size.

2) Memory Efficiency: The minimizer approach achieved significant memory reduction:

- Total unique minimizers: 4,515,771 (vs. 22,104,664 k-mers)
- Overall reduction ratio: 0.20 (80% reduction in index size)
- Serialized index size: 206.75 MB (vs. 1,012.05 MB for full k-mer index)

This 5-fold reduction in memory requirements enables processing of much larger reference databases on resource-constrained systems.

3) Classification Accuracy: Despite the substantial reduction in index size, the minimizer-based classification maintained—and in some cases improved—accuracy:

a) Error-free reads: For reads without sequencing errors, the minimizer approach achieved identical classification results to the full k-mer index with 0% difference across all categories:

- E. coli K-12 MG1655: 30.45% (identical to full k-mer)
- B. subtilis 168: 5.05% (identical to full k-mer)
- P. aeruginosa PAO1: 5.03% (identical to full k-mer)
- S. aureus NCTC 8325: 5.03% (identical to full k-mer)
- M. tuberculosis H37Rv: 5.06% (identical to full k-mer)
- Multi-match and unclassified rates also remained identical

b) Error-containing reads:: Surprisingly, the minimizer approach performed better on error-containing reads:

- E. coli K-12 MG1655: 2.91% (vs. 2.15% with full k-mer, +0.76%)
- B. subtilis 168: 0.59% (vs. 0.48% with full k-mer, +0.11%)
- P. aeruginosa PAO1: 0.56% (vs. 0.35% with full k-mer, +0.21
- S. aureus NCTC 8325: 0.49% (vs. 0.36% with full k-mer, +0.13%)
- M. tuberculosis H37Rv: 0.51% (vs. 0.42% with full k-mer, +0.09%)
- Unclassified reads: 94.93% (vs. 96.23% with full k-mer, -1.30%)

The improved classification rate for error-containing reads suggests that minimizers offer greater robustness to sequencing errors. This may be because minimizers effectively "sample" k-mers from each window, reducing the impact of isolated sequencing errors while preserving the overall genomic signature.

4) Performance Trade-offs: The minimizer approach presents a time-versus-space trade-off:

- Index construction: Minimizers required more time (53-59 seconds vs. 17-18 seconds for full k-mer index)
- Classification speed: Minimizers took slightly longer (1.98 vs. 1.50 seconds for error-free reads, 5.43 vs. 2.99 seconds for error-containing reads)
- Memory usage: Minimizers used only 20% of the memory compared to the full k-mer index

For most applications, especially those involving large reference databases or resource-constrained environments, the minor increase in processing time is an acceptable trade-off for the substantial memory savings and improved error tolerance

V. Real-world data and tools

A. Comparison with Kraken2

To benchmark my implementations against industry-standard tools, I evaluated Kraken2, a widely-used metagenomic classifier. I built a custom Kraken2 database using the same five reference genomes and configured parameters to perform exact-like matching: maximum k-mer length of 35 bp, disabled minimizer spaces, high confidence threshold (1.0), and quick mode for stringent matching. Reads were processed individually rather than in paired mode to enable direct comparison with my implementations.

1) Classification Results: Table IV presents the classification results from Kraken2 for both error-free and error-containing reads.

TABLE IV: Classification Results for Kraken2

Classification	Error-free(%)	Error-containing(%)
E. coli	59.10	53.84
B. subtilis	9.54	8.58
P. aeruginosa	9.92	9.12
S. aureus	9.44	8.76
M. tuberculosis	9.76	9.20
Reads with multiple matches	0.00	0.00
Unclassified	2.00	10.26

The most striking observation is Kraken2's dramatically higher classification rate, particularly for error-containing reads. While my best implementation classified just 5.07% of error-containing reads, Kraken2 successfully classified 92.10%. For error-free reads, Kraken2 achieved a remarkable 98.18% classification rate compared to approximately 50.7% with my approaches.

2) Performance Metrics: Kraken2 also demonstrated superior computational efficiency:

- Database building time: 12.62 seconds (vs. 44.14 seconds for my k-mer index)
- Peak memory during database building: 17.02 MB (vs. 1,364.47 MB)
- Classification time: 0.09-0.18 seconds (vs. 0.73-15.06 seconds)
- Peak memory during classification: 55-70 MB (vs. 76-439 MB)

While Kraken2 required more memory during database construction, it achieved over 100x speedup during classification with lower memory usage, demonstrating exceptional performance optimization.

3) Analysis of Performance Gap: The performance disparity can be attributed to several factors in Kraken2's sophisticated design:

1) **Taxonomically-aware approach:** Kraken2 leverages the hierarchical taxonomic relationships between organisms
2) **Weighted classification scheme:** K-mers are assigned weights based on their specificity in the taxonomic tree
3) **Partial k-mer matching:** Kraken2 can classify reads based on partial k-mer matches, providing robustness to sequencing errors
4) **Bi-directional strand matching:** Unlike my implementations which were restricted to forward strand matching only, Kraken2 considers both strands
5) **Highly optimized data structures:** Years of development have produced exceptionally efficient algorithms

This comparison highlights the sophistication of purpose-built bioinformatics tools and illustrates the challenges of developing efficient metagenomic classifiers.

B. Real-world use case

To evaluate the practical utility of metagenomic classification in environmental monitoring, I applied Kraken2 with the Standard-8 pre-built database to analyze real-world samples from two distinct environments: human gut microbiomes and wastewater.

1) Methodology: The analysis pipeline to process Kraken2 classification contained results from 10 metagenomic samples: 5 from human gut (SRR11412*) and 5 from wastewater treatment plants (SRR21907*). The pipeline implemented the following key functions:

- **parse_kraken _report():** Extracted taxonomic information from Kraken2 reports, filtering by abundance threshold (0.1%) and taxonomic level. For each taxon, it recorded the percentage abundance, taxonomic rank code, and cleaned taxon name.
- **create_taxonomic_profile_matrix():** Built a samples-by-taxa matrix where each cell contained the abundance percentage of a particular genus in a sample.
- **plot_pca():** Performed Principal Component Analysis after standardizing the data to identify main sources of variation between samples.
- **perform_hierarchical_clustering():** Applied hierarchical clustering using Ward's method to group samples based on taxonomic similarity.

2) Results and Analysis: The analysis revealed distinct taxonomic profiles between gut and wastewater samples:

a) Environmental Separation: All analysis methods successfully separated the samples into two distinct classes that perfectly corresponded to their environmental origins.

The hierarchical clustering analysis (Figure 2) showed a perfect separation between human gut and wastewater samples at the highest level of the dendrogram. The large distance

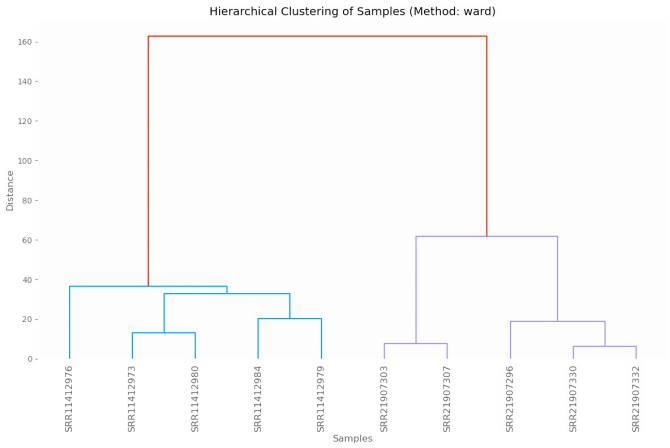

Fig. 2: Hierarchical clustering dendrogram using Ward's method. The dendrogram shows clear separation of samples into two distinct clusters corresponding to their environmental sources: human gut samples (SRR11412*) on the left and wastewater samples (SRR21907*) on the right. The large distance between these clusters highlights the substantial differences in microbial community composition between the two environments.

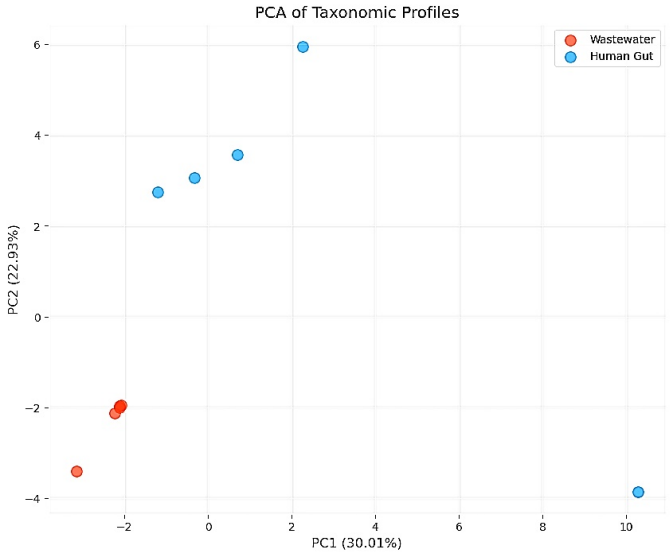

Fig. 3: PCA of taxonomic profiles showing clear separation between human gut samples (blue) and wastewater samples (red). The first two principal components explain 52.94% of the total variance. The complete separation of samples along both axes demonstrates that taxonomic composition is strongly correlated with environmental source.

between the two main clusters (approximately 160 units) indicates substantial differences in their microbial compositions, with human gut samples (SRR11412*) forming a distinct cluster from wastewater samples (SRR21907*).

Principal Component Analysis (Figure 3) showed complete separation of samples along both PC1 (30.01% variance) and

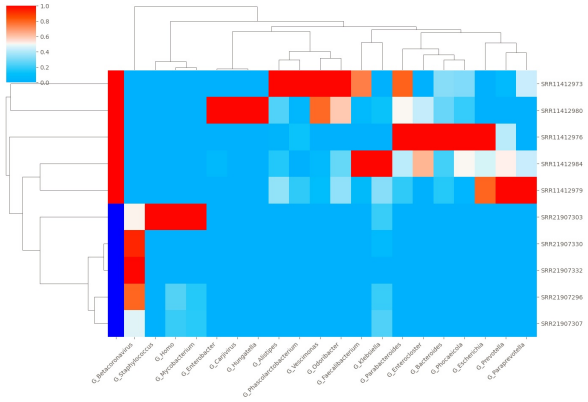

Fig. 4: Clustering of samples by taxonomic abundance. The heatmap shows genus-level abundance (standardized) across all samples. Row colors indicate sample source: red for human gut (SRR11412*) and blue for wastewater (SRR21907*) samples. The clear clustering by environment and distinctive genus abundance patterns highlight the environment-specific microbial communities.

PC2 (22.93% variance). The human gut samples (blue) and wastewater samples (red) form distinct clusters in the PCA plot, with no overlap between them. This strong separation in the first two principal components (explaining over 50% of variance) confirms highly environment-specific taxonomic profiles.

The taxonomic abundance heatmap (Figure 4) further illustrates the distinctive microbial composition patterns between the two environments. The clustering of samples by environment (shown in the dendrogram at the top) and the distinct abundance patterns (red for high abundance, blue for low abundance) reveal environment-specific microbial signatures.

This clear separation reflects fundamental biological differences between these environments: human gut microbiomes are dominated by specialized anaerobes adapted for the gut environment, while wastewater samples show higher abundance of environmentally resilient bacteria and potential pathogens. The real-world application demonstrates the utility of metagenomic classification for environmental monitoring and microbial community characterization. The ability to differentiate samples based on their source environments highlights the power of taxonomic profiling for ecological and public health applications.

VI. Conclusion

This project explored various approaches to metagenomic sequence classification, from fundamental string matching algorithms to k-mer indexing with minimizers, and compared these custom implementations with industry-standard tools like BLAST and Kraken2.

A. Key Findings and Insights

Our investigation revealed several important insights:

1) **Algorithmic Trade-offs:** String matching and k-mer indexing achieved comparable classification accuracy for error-free reads ($\sim$50.7%), but k-mer indexing offered significantly better performance (0.73 seconds vs. several minutes) and scalability.
2) **Error Tolerance:** All exact matching approaches showed poor resilience to sequencing errors, with classification rates dropping dramatically from $\sim$50% for error-free reads to less than 4% for error-containing reads. Purpose-built tools like BLAST and Kraken2 employ sophisticated error-handling mechanisms that significantly outperform naive approximate matching.
3) **Memory Efficiency:** The minimizer scheme reduced index size by 80% while maintaining comparable classification accuracy, and even improved classification for error-containing reads, demonstrating its value for resource-constrained applications.
4) **Strand Limitations:** Our implementations were restricted to forward-strand matching only, unlike Kraken2 which considers both strands. This limitation significantly impacts real-world applications, as DNA fragments can come from either strand.
5) **Performance Gap:** Specialized bioinformatics tools significantly outperformed custom implementations, with Kraken2 achieving $\sim$ 98% classification for error-free reads and $\sim$ 90% for error-containing reads, along with faster classification speeds.
6) **Classification Strategies:** Our implementation handled multi-matching reads by retaining all matching organisms, which differs from the LCA (Lowest Common Ancestor) approach used by tools like Kraken2.

B. Technical Insights

The implementation experience yielded valuable technical lessons:

- **Data Structure Importance:** The choice of data structures dramatically impacts performance and memory usage. Hash-based structures provided critical O(1) lookup performance for k-mer indexing.
- **Biological Context:** Understanding biological constraints (e.g., k-mer distribution patterns in real genomes) helps explain unexpected observations like the vast difference between theoretical and actual k-mer counts.
- **Space-Time Tradeoffs:** Every optimization involves tradeoffs. The minimizer approach traded slightly longer processing times for drastically reduced memory requirements while maintaining accuracy.
- **Domain Knowledge:** Specialized tools incorporate biological domain knowledge that significantly enhances performance, highlighting the importance of understanding the underlying properties of genomic data.

C. Real-world Applications

The environmental sample analysis demonstrated the practical utility of these methods. Using Kraken2's taxonomic profiling capabilities, we could clearly distinguish between human gut and wastewater samples based on their distinctive microbial communities. The identification of environment-specific taxa and significant differences in alpha diversity between environments validates the effectiveness of metagenomic classification for environmental monitoring.

D. Future Directions

Based on our findings, several promising directions for improvement emerge:

1) **Hybrid Approaches:** Combining the memory efficiency of minimizers with more sophisticated error-tolerance mechanisms could yield better performance on real sequencing data.
2) **Taxonomic Awareness:** Incorporating hierarchical taxonomic relationships into the classification strategy could improve accuracy and interpretability.
3) **Bi-directional Matching:** Extending implementations to consider both DNA strands would improve real-world applicability.
4) **Parallelization and Hardware Optimization:** Many aspects of the implementation could benefit from further parallelization and specialized hardware accelerators (e.g., GPUs).
5) **Advanced Minimizer Schemes:** Exploring more sophisticated minimizer selection schemes could further improve the space-time trade-off.
6) **Larger-scale Environmental Studies:** Expanding the environmental analysis to include more diverse sample types could yield insights into microbial adaptation across different ecological niches.

In conclusion, this project demonstrated the fundamental principles and challenges of metagenomic classification while highlighting the value of algorithmic optimizations like minimizers. The significant performance gap between custom implementations and specialized tools reflects the sophisticated engineering that has gone into these bioinformatics applications, but also presents opportunities for further innovation in this rapidly evolving field.

REFERENCES

[1] Repository Link for Implementation https://github.com/omar404ahmed/bioinformatics_249/tree/main/one

[2] Wood, D. E., Lu, J., & Langmead, B. (2019). *Improved metagenomic analysis with Kraken 2*. Genome Biology, 20(1), 257. https://doi.org/10.1186/s13059-019-1891-0

[3] Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M., & Yorke, J. A. (2004). *Reducing storage requirements for biological sequence comparison*. Bioinformatics, 20(18), 3363-3369. https://doi.org/10.1093/bioinformatics/bth408

[4] Jain, C., Rhie, A., Hansen, N. F., Koren, S., & Phillippy, A. M. (2022). *Long-read mapping to repetitive reference sequences using Winnowmap2*. Nature Methods, 19(6), 705-710. https://doi.org/10.1038/s41592-022-01457-8

[5] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). *Basic local alignment search tool*. Journal of Molecular Biology, 215(3), 403-410. https://doi.org/10.1016/S0022-2836(05)80360-2

[6] Breitwieser, F. P., Baker, D. N., & Salzberg, S. L. (2018). *KrakenUniq: confident and fast metagenomics classification using unique k-mer counts*. Genome Biology, 19(1), 198. https://doi.org/10.1186/s13059-018-1568-0

[7] Le, V. V., Lang, T., & Binh, N. (2024). *Minimizers in Bioinformatics*. Genome Biology, 25, Article 36. https://genomebiology.biomedcentral.com/articles/10.1186/s13059-024-03414-4

[8] Li, H. (2018). *Minimap2: pairwise alignment for nucleotide sequences*. Bioinformatics, 34(18), 3094-3100. https://doi.org/10.1093/bioinformatics/bty191

[9] Manekar, S. C., & Sathe, S. R. (2018). *A benchmark study of k-mer counting methods for high-throughput sequencing*. GigaScience, 7(12), giy125. https://doi.org/10.1093/gigascience/giy125

[10] Marçais, G., Pellow, D., Bray, D., Hernaez, M., & Möhle, M. (2017). *Improving the performance of minimizers and winnowing schemes*. Bioinformatics, 33(14), i110-i117. https://doi.org/10.1093/bioinformatics/btx235