Bioinformatics Algorithms: Implementation and Evaluation of Genome Assembly Methods

Omar Ahmed
MS in Bioengineering
KAUST ID: 212769
CS 249: Assignment 2

***Abstract*—This paper presents the implementation and evaluation of two fundamental genome assembly algorithms: De Bruijn Graph (DBG) and Overlap-Layout-Consensus (OLC). We implement these approaches from scratch and evaluate them on synthetic and real datasets. The DBG approach builds a graph where nodes represent k-mers and paths through the graph represent potential contigs, while the OLC approach identifies overlaps between reads, constructs an overlap graph, and generates consensus sequences. Our evaluation reveals that the DBG algorithm excels with error-free data of any read length but deteriorates dramatically with error-containing reads, while OLC demonstrates superior error tolerance, particularly for long-read data. We demonstrate how k-mer size selection significantly impacts assembly completeness and contiguity, with larger k-mers (k=47) achieving perfect reference coverage compared to smaller k-mers (k=37) in our test case. Comparison with professional tools like SPAdes highlights the critical importance of error correction in real-world assembly applications. The paper also presents a high-quality de novo assembly of the Scincus mitranus (sandfish lizard) genome using Verkko, achieving a 3.49 Gb assembly with an N50 of 2.04 Mb and exceptional base-level accuracy (QV 42-44), demonstrating the effectiveness of hybrid assembly approaches combining PacBio HiFi and Oxford Nanopore reads.**

***Index Terms*—genome assembly, de Bruijn graph, overlap-layout-consensus, bioinformatics algorithms, next-generation sequencing**

I. Introduction

Genome assembly is a critical process in bioinformatics that reconstructs complete genome sequences from fragmented DNA sequence "reads." This process is fundamental to many applications in genomics, including variant discovery, comparative genomics, and functional annotation. This paper focuses on implementing and evaluating two fundamental genome assembly approaches: De Bruijn Graph (DBG) and Overlap-Layout-Consensus (OLC).

The advent of next-generation sequencing technologies has led to an explosion of sequencing data, making efficient and accurate assembly algorithms essential. While commercial and open-source assemblers are available, understanding the underlying algorithms through implementation provides valuable insights into their strengths, limitations, and potential optimizations.

We implemented both DBG and OLC approaches from scratch and evaluated them on synthetic datasets and the MERS-CoV genome. For comparison, we also used established assemblers like Velvet. Our evaluation includes standard assembly metrics, graph visualization, and analysis of parameter effects.

The remainder of this paper is organized as follows: Section 2 details our implementation of the De Bruijn Graph and Overlap-Layout-Consensus assembly algorithms, describing their key components and potential improvements. Section 3 presents our experimental evaluation, including verification against established tools, visualization and analysis of assembly graphs, investigation of k-mer size effects, comparative analysis of algorithm performance on MERS virus data with varying error profiles, and benchmarking against professional assemblers. Section 4 describes our de novo assembly of the Scincus mitranus genome, including assembly approaches, comprehensive quality evaluation, and biological significance. Section 5 concludes the paper with a summary of our findings and directions for future work. Throughout our analysis, we demonstrate how algorithm selection and parameter optimization impact assembly quality, while highlighting the importance of error correction and the complementary strengths of different assembly approaches.

II. Implementation of Basic Genome Assembly Algorithms

A. De Bruijn Graph (DBG) Assembly

1) Algorithm Implementation: The De Bruijn Graph (DBG) assembly algorithm builds a graph representation of sequence data where nodes represent (k-1)-mers and edges represent k-mers. This approach is particularly effective for handling short reads with high coverage, as commonly produced by Illumina sequencing platforms. The implementation takes FASTQ files as input, constructs the de Bruijn graph, identifies Eulerian paths to form contigs, and outputs the assembled sequences as a FASTA file.

The algorithm follows these key steps:

1) Parse input reads from FASTQ files
2) Break each read into overlapping k-mers
3) Construct the de Bruijn graph:
   - Each node is a (k-1)-mer (prefix or suffix of a k-mer)
   - Each edge represents a k-mer, connecting its prefix and suffix nodes
   - Track edge coverage information (how many times a k-mer appears)

4) Find Eulerian paths in the graph using Hierholzer's algorithm
5) Convert the paths to contig sequences
6) Output contigs to FASTA format and optionally export the graph to GFA format

2) Key Components: The implementation uses several key data structures and algorithms:

a) Graph Representation: The de Bruijn graph is implemented using NetworkX's `MultiDiGraph`, allowing for multiple directed edges between the same pair of nodes. Nodes represent (k-1)-mers, while edges represent k-mers. Each edge stores:

- An edge ID for reference
- The sequence (last base of the k-mer)
- Coverage information (frequency of occurrence)

b) Eulerian Path Finding: Hierholzer's algorithm is implemented to find Eulerian paths in the graph, which represent potential contigs. The algorithm:

- Prioritizes nodes with unbalanced in/out-degree as starting points
- Handles disconnected components in the graph
- Constructs paths by traversing and removing edges

c) Contig Construction: Paths found by Hierholzer's algorithm are converted to contig sequences by:

- Starting with the first (k-1)-mer in the path
- Extending by adding the last base of each subsequent k-mer
- Creating SeqRecord objects for output

d) Assembly Metrics: The implementation calculates several reference-free assembly metrics:

- Total assembly length
- Number of contigs
- GC content percentage
- Largest contig length
- N50 and N90 statistics
- L50 (number of contigs required to reach 50% of total assembly length)

e) Potential Improvements: While the current implementation provides a functional DBG assembler, several enhancements could improve its performance and accuracy:

- Error correction for noisy reads
- Coverage-based filtering to remove low-coverage k-mers (likely errors)
- Tip removal algorithm to handle sequencing errors at read ends
- Bubble removal algorithm to handle sequence variants
- More sophisticated contig extension strategies
- Multithreading for faster processing of large datasets

B. Overlap-Layout-Consensus (OLC) Assembly

1) Algorithm Implementation: The Overlap-Layout-Consensus (OLC) assembly algorithm is particularly effective for longer reads where overlaps between reads can be more confidently identified. This implementation takes FASTQ files as input, identifies overlaps between reads, constructs an overlap graph, finds non-branching paths, generates a layout of reads, and computes a consensus sequence for each contig.

The algorithm follows these key steps:

1) Parse input reads from FASTQ files, including quality scores
2) Identify and filter out contained reads (reads entirely contained within other reads)
3) Compute read overlaps using an efficient two-phase approach:
   - Initial filtering using MinHash sketches to estimate Jaccard similarity
   - Precise suffix-prefix overlap calculation for promising candidates
4) Construct an overlap graph where:
   - Nodes represent reads
   - Directed edges represent suffix-prefix overlaps between reads
   - Edge weights incorporate both overlap length and sequence identity
5) Find paths through the graph using a greedy approach
6) Generate a layout by positioning reads along each path
7) Compute a quality-aware consensus sequence for each layout
8) Output contigs to FASTA format

2) Key Components: The implementation uses several sophisticated data structures and algorithms to achieve efficient and accurate assembly:

a) MinHash Sketching: A dimensionality reduction technique used to quickly estimate sequence similarity:

- Extracts k-mers from each read
- Hashes each k-mer and keeps only the smallest hash values (the "sketch")
- Estimates Jaccard similarity between reads by comparing their sketches
- Reduces the computational complexity from $O(n^2)$ to $O(n \cdot \text{sketch\_size})$

b) Overlap Detection: A two-phase approach that balances efficiency and accuracy:

- First phase: Filter candidate pairs using MinHash Jaccard similarity
- Second phase: Compute exact suffix-prefix overlaps for promising candidates
- Parameters control minimum overlap length and identity requirements

c) Containment Detection: Identification of reads fully contained within other reads:

- Reduces graph complexity by filtering redundant information
- Implemented in parallel for efficiency on large datasets

d) Quality-Aware Consensus: Generation of consensus sequences that leverage quality scores:

- Converts Phred quality scores to error probabilities

- Uses weighted voting at each position based on base quality
- Produces more accurate contigs by prioritizing high-quality base calls

e) Parallelization: The implementation utilizes parallel processing for computationally intensive steps:

- MinHash sketch generation
- Containment detection
- Overlap computation

f) Potential Improvements: While the current implementation provides an efficient and accurate OLC assembler, several enhancements could further improve its performance:

- **Advanced Overlap Detection**: Implement suffix arrays or FM-index for more efficient overlap detection; add support for approximate matching to better handle sequencing errors
- **Graph Simplification**: Implement bubble detection and resolution for handling polymorphisms; add transitive reduction to simplify the overlap graph
- **Consensus Improvements**: Implement a multiple sequence alignment approach for consensus generation; add error correction during consensus calculation
- **Resource Optimization**: Add disk-based solutions for handling very large datasets; implement better memory management for the overlap graph
- **Algorithm Enhancements**: Implement more sophisticated path finding algorithms; add repeat resolution strategies
- **Scaffolding**: Incorporate paired-end information when available to resolve complex regions and extend contigs

III. Applications and Evaluation

A. Verification of De Bruijn Graph Implementation

To validate the correctness and efficiency of our De Bruijn Graph (DBG) implementation, we compared it with Velvet, a well-established DBG-based genome assembler. Both assemblers were run on the same dataset (reads_r.fastq) with similar k-mer parameters, and the resulting assemblies were evaluated using QUAST with reference_r.fasta as the reference genome.

1) QUAST Comparison Results: Figure **??** shows the QUAST evaluation metrics comparing our DBG implementation (k=37) with Velvet's output.

The key observations from the QUAST metrics are:

- **Reference Coverage**: Our implementation achieved a 90% genome fraction compared to Velvet's 88.08%, indicating slightly better coverage of the reference genome.
- **Contig Quality**: Both assemblers produced single contigs with similar lengths—936 bp for our implementation versus 916 bp for Velvet. Since the reference genome is 1040 bp, both contigs represent over 88% of the complete sequence.
- **Contig Accuracy**: Both assemblies showed perfect accuracy with no misassemblies, mismatches, or indels, demonstrating the reliability of both methods for this dataset.
- **Structural Integrity**: Both implementations achieved a duplication ratio of 1, indicating no duplicated regions in the assembly.

Figure **??** shows the alignment of both contigs to the reference genome, confirming the high quality of the assemblies.

The cumulative length plot in Figure **??** provides another perspective on the assemblies, showing that both implementations yield contigs of similar quality, with our implementation (blue line) recovering slightly more of the reference sequence than Velvet (red line).

2) Conclusions from Verification: Our De Bruijn Graph implementation performs comparably to Velvet, a widely used genome assembler, and actually achieves slightly better reference coverage. This confirms the correctness of our implementation and demonstrates its effectiveness for genome assembly tasks. The results validate our approach to graph construction, path finding, and consensus generation.

Key advantages observed in our implementation:

- Slightly better genome fraction (90% vs. 88.08%)
- Longer contigs (936 bp vs. 916 bp)
- Comparable accuracy (zero errors in both)

This verification establishes a solid foundation for further applications and optimizations of our assembler.

B. Visualization and Analysis of Assembly Graphs

To understand the structural features of de Bruijn graphs and gain insights that can improve assembly quality, we constructed an assembly graph for the reads_b.fastq dataset with k=40 and visualized it using Bandage.

1) Graph Construction and Initial Metrics: The de Bruijn graph was constructed with k=40 using our DBG implementation, which produced the following assembly metrics:

Metric	Value
Total assembly length	1,079 bp
Number of contigs	2
GC content	52.27%
Largest contig	749 bp
N50	749 bp
N90	330 bp
L50	1

TABLE I: Assembly metrics for reads_b.fastq with k=40

2) Bandage Visualization Analysis: The de Bruijn graph was exported in GFA format and visualized using Bandage, as shown in Figure **??**.

3) Observations and Insights: Analysis of the de Bruijn graph visualization reveals several important features:

1) **Circular Structure**: The primary component of the graph forms a near-complete circle, suggesting that reads_b.fastq may represent a circular genome (such as a plasmid, mitochondrial DNA, or viral genome). This circular structure explains why we obtained two contigs rather than one—the assembly algorithm broke the circle at some point.
2) **Small Bubble**: A small "bubble" structure is visible in the graph, where the path briefly splits into two

alternative paths before rejoining. This likely represents either:

- A sequencing error in some reads
- A genuine small-scale variant (SNP or small indel)
- A short repeat region with similar but not identical sequences

3) **Contig Junction**: The junction between the two contigs is clearly visible, showing where the assembler was unable to resolve the complete circular path. This break point may correspond to a region with lower coverage or higher error rates.
4) **Uniform Coverage**: The relatively uniform thickness of the edges throughout most of the graph indicates consistent read coverage across the genome, with the exception of the bubble region where coverage is divided between alternative paths.

4) Improving Assembly Using Graph Insights: Based on the graph visualization, several strategies can be employed to improve the assembly:

1) **Circle Completion**: Since the graph strongly suggests a circular genome, we can modify our contig-generation algorithm to specifically look for and connect circular paths, potentially producing a single complete contig.
2) **Bubble Resolution**: Implementing a bubble detection and resolution algorithm would allow us to identify the most likely correct path through the bubble based on coverage or quality scores, thereby improving assembly accuracy.
3) **Junction Analysis**: Examining the sequence characteristics at the junction between the two contigs might reveal why the assembler broke the path there. Adding specific handling for problematic sequence motifs (like homopolymers or short repeats) could help resolve such junctions.
4) **Coverage-Based Filtering**: Adding coverage-based filtering to remove or correct low-coverage paths could eliminate spurious branches and improve graph simplicity.
5) **Alternative k-mer Sizes**: Since the graph structure is relatively simple but still contains one unresolved feature (the bubble), experimenting with different k-mer sizes might resolve this issue—larger k-mers could span the bubble entirely, while smaller k-mers might provide higher resolution of the variant.

The Bandage visualization proves invaluable for understanding the underlying structure of the genomic data and offers clear directions for improving assembly algorithms. The circular nature of this particular genome would have been difficult to identify from contig sequences alone, demonstrating the value of graph-based analysis in genome assembly.

C. Effect of k-mer Size on De Bruijn Graph Assembly

To investigate the effect of k-mer size on assembly quality and characteristics, we applied our DBG implementation to reads_r.fastq with two different k-mer sizes: k=37 (lower than the typical read length) and k=47 (higher than the typical read length). The assemblies were evaluated using QUAST against the reference genome (reference_r.fasta).

1) Assembly Metrics Comparison: The key assembly and reference-based metrics for both k-mer sizes are presented in Table II.

Metric	k=37	k=47
Total length	936 bp	1,040 bp
Number of contigs	1*	1
GC content	51.50%	51.25%
Reference length	1,040 bp	1,040 bp
Genome fraction (%)	90.000%	100.000%
Largest contig	936 bp	1,040 bp
N50	936 bp	1,040 bp
Misassemblies	0	0
Mismatches per 100 kbp	0.00	0.00
Indels per 100 kbp	0.00	0.00
Duplication ratio	1.000	1.000

TABLE II: QUAST comparison of assemblies for reads_r.fastq with different k-mer sizes. *Note: The k=37 assembly actually produced 2 contigs, but only 1 was 500 bp (QUAST's default threshold for most statistics).

2) Assembly Graph Visualizations: The de Bruijn graphs for both assemblies were exported to GFA format and visualized using Bandage, as shown in Figure **??**.

3) QUAST Alignment Visualizations: The alignment of the assembled contigs to the reference genome provides additional insights into assembly quality, as shown in Figures **??** and **??**.

4) Analysis of Differences: The QUAST evaluation and graph visualizations reveal several key insights about the impact of k-mer size on de Bruijn graph assembly:

1) **Genome Completeness**: The k=47 assembly achieved 100% genome coverage, perfectly matching the reference length of 1,040 bp. In contrast, the k=37 assembly covered only 90% of the reference genome, with its largest contig being 936 bp.
2) **Contig Continuity**: With k=47, the assembler produced a single contig covering the entire genome, whereas with k=37, the assembly was fragmented into two contigs (although only the larger 936 bp contig met QUAST's minimum threshold of 500 bp). This demonstrates how a larger k-mer size can better resolve challenging regions in the genome.
3) **Assembly Accuracy**: Both assemblies showed perfect accuracy with zero misassemblies, mismatches, or indels. This indicates high-quality reads with few sequencing errors, which allowed even the larger k-mer size to perform well despite typically being more sensitive to errors.
4) **GC Content**: The k=47 assembly exactly matched the reference GC content (51.25%), while the k=37 assembly showed a slight deviation (51.50%). This minor difference likely reflects the incomplete nature of the k=37 assembly.
5) **Graph Topology**: The k=37 graph shows a linear structure with a break, while the k=47 graph shows a

complete path with a small bubble. The k=47 assembler successfully resolved this bubble to produce a complete contig, likely by selecting the higher-coverage path through the bubble region.

5) Explaining the Graph Differences: The key difference in the assembly graphs can be understood by examining how each k-mer size handles specific genomic features:

- With k=37, the assembler encountered a region that it could not confidently traverse, resulting in a graph break and consequently two separate contigs. This could be due to:
  - A repetitive element longer than 37 bp
  - A low-coverage region where some k-mers were not detected
  - A complex sequence motif creating ambiguity in the graph
- With k=47, the larger k-mer size was able to:
  - Span the problematic region that caused the break in the k=37 assembly
  - Capture enough unique context to resolve the potential repeat
  - Create a bubble where alternative paths could be evaluated and the best path selected

6) Optimal K-mer Size Selection: For this specific dataset, k=47 clearly produced a superior assembly with perfect reference coverage and a single complete contig. This outcome demonstrates several principles for selecting optimal k-mer sizes:

- **Read Length Consideration**: The optimal k-mer size is often related to read length. Here, k=47 was successful despite being relatively large compared to typical short-read lengths.
- **Error Rate Influence**: The success of k=47 suggests the error rate in this dataset is low. For datasets with higher error rates, smaller k-mers might perform better despite having less power to resolve repeats.
- **Repeat Resolution**: Larger k-mers better resolve repeats by capturing more unique context. The k=47 assembly successfully captured the entire genome without breaks, suggesting it effectively spanned all repetitive or complex regions.
- **Coverage Depth Impact**: Sufficient coverage depth is crucial for larger k-mers to be effective. The reads_r.fastq dataset likely has high coverage, allowing even larger k-mers to have adequate sampling.

7) Conclusions: This comparison clearly demonstrates how k-mer size selection can dramatically affect assembly outcomes, even for a relatively simple genome. The k=47 assembly achieved perfect coverage and contiguity, while the k=37 assembly missed approximately 10% of the genome.

This finding highlights the importance of experimenting with different k-mer sizes during de Bruijn graph assembly. While no single k-mer size is optimal for all datasets, larger k-mers tend to perform better for high-quality data with sufficient coverage, particularly when resolving repeats is a primary concern.

For practical applications, the results suggest a strategy of generating multiple assemblies with different k-mer sizes and selecting the best result based on reference-free assembly metrics when a reference is unavailable, or using reference-based metrics like genome fraction when a reference is available.

D. MERS Virus Assembly

To evaluate the performance of our assembly algorithms on more realistic data, we applied both the DBG and OLC implementations to the Middle East respiratory syndrome-related coronavirus (MERS-CoV) genome. This analysis allowed us to assess how each algorithm handles different read types (HiSeq short reads vs. ONT long reads) and how sequencing errors affect assembly quality.

1) Experimental Setup: We conducted experiments with the following datasets:

- **Error-free Illumina HiSeq reads**: Simulated short reads without errors
- **Error-free Oxford Nanopore (ONT) reads**: Simulated long reads without errors
- **Error-containing Illumina HiSeq reads**: Realistic short reads with simulated errors
- **Error-containing Oxford Nanopore reads**: Realistic long reads with simulated errors

For the DBG algorithm, we used a k-mer size of 31 for all assemblies. For the OLC algorithm, parameters varied based on read type and error status:

- **Error-free reads**: minimum overlap length of 30 bp, minimum identity of 0.95
- **Error-containing HiSeq**: minimum overlap length of 30 bp, minimum identity of 0.90
- **Error-containing ONT**: "ultra-permissible" parameters with minimum overlap length of 15 bp, minimum identity of 0.75, reduced k-mer size for sketching (8), and increased sketch size (250)

2) Results Overview: Table III presents the key metrics for all MERS virus assemblies, and Figure **??** shows a heatmap visualization of the most critical assembly metrics.

3) Error-free Read Assembly:

a) De Bruijn Graph Performance: The DBG algorithm excelled on error-free data, regardless of read type. Both HiSeq and ONT error-free assemblies achieved near-perfect results:

- **Completeness**: Both covered almost the entire reference genome (97.9% for HiSeq and 98.8% for ONT).
- **Contiguity**: Each produced exactly one contig closely matching the reference length of 30,119 bp.
- **Accuracy**: Zero mismatches, indels, or misassemblies.
- **GC Content**: Accurate representation of reference GC content (41.26-41.27% vs. 41.24% reference).

b) OLC Performance: The OLC algorithm showed variable performance on error-free data depending on read type:

- **ONT Reads**: Excellent genome coverage (97.6%) but with significant duplication (3.95x) and fragmentation (6 contigs), suggesting difficulty with repeat resolution.

TABLE III: QUAST evaluation metrics for MERS virus genome assemblies

Metric	De Bruijn Graph (k=31)				Overlap-Layout-Consensus			
	Error-free HiSeq	**Error-free ONT**	**Error HiSeq**	**Error ONT**	**Error-free HiSeq**	**Error-free ONT**	**Error HiSeq**	**Error ONT**
Total length	29,482 bp	29,748 bp	44,837 bp	649,969 bp	122,061 bp	116,134 bp	131,397 bp	74,432 bp
# contigs (500 bp)	1	1	8	9	32	6	15	4
Largest contig	29,482 bp	29,748 bp	18,800 bp	295,326 bp	1,172 bp	25,031 bp	833 bp	20,731 bp
N50	29,482 bp	29,748 bp	6,077 bp	206,097 bp	578 bp	23,922 bp	560 bp	19,077 bp
Genome fraction	97.9%	98.8%	70.8%	18.2%	55.1%	97.6%	26.2%	86.5%
Duplication ratio	1.000	1.000	1.008	1.282	1.228	3.949	1.157	2.855
Mismatches/100 kbp	0.00	0.00	5,587.09	554.21	0.00	0.00	43.88	298.39
Indels/100 kbp	0.00	0.00	669.89	2,231.06	0.00	0.00	0.00	1,149.19
# misassemblies	0	0	0	0	0	0	0	0
# local misassemblies	0	0	20	0	0	0	0	0
# unaligned contigs	0	0	2+4 part	2+6 part	0	0	0	0
Unaligned length	0	0	15,507 bp	630,774 bp	0	0	0	0

- **HiSeq Reads**: Poor genome coverage (55.1%) with extreme fragmentation (32 contigs 500 bp, 461 total contigs), indicating that short overlaps between short reads were insufficient for effective assembly.

c) Comparative Analysis: For error-free data, our findings demonstrate:

- DBG was superior for short reads (HiSeq), achieving higher genome coverage with a single contig versus OLC's highly fragmented assembly.
- Both algorithms performed well with error-free ONT long reads, but DBG produced a more concise and accurate assembly without duplication.
- The DBG approach shows remarkable versatility, performing well with both read types when errors are absent.

4) Error-containing Read Assembly:

a) De Bruijn Graph Performance: The introduction of sequencing errors dramatically impacted DBG assemblies, with severe degradation in quality:

- **HiSeq Reads**: Moderate genome coverage (70.8%) was achieved, but with concerning fragmentation (8 contigs 500 bp) and an extremely high mismatch rate (5,587 mismatches per 100 kbp) - over 100 times higher than the OLC approach.
- **ONT Reads**: Catastrophic performance with only 18.2% genome coverage despite massive overassembly (650 kbp total vs. 30 kbp reference). Most of the assembly (631 kbp, 97%) could not be aligned to the reference, indicating extensive assembly errors.

b) OLC Performance: The OLC algorithm demonstrated superior handling of error-containing reads:

- **HiSeq Reads**: Lower genome coverage (26.2%) but with substantially higher accuracy - only 43.88 mismatches per 100 kbp and zero indels, compared to thousands of errors in the DBG assembly.
- **ONT Reads**: Remarkable performance with ultra-permissible parameters, achieving 86.5% genome coverage with just 4 contigs. Despite some duplication (2.86x) and moderate error rates, the OLC assembly captured most of the genome with acceptable accuracy.

c) Parameter Sensitivity in OLC: For error-containing ONT reads, standard OLC parameters failed completely. We systematically tested increasingly permissive parameters until achieving successful assembly:

- Standard parameters (30 bp overlap, 0.9 identity): No contigs
- Relaxed parameters (20 bp overlap, 0.8 identity): No contigs
- Modified k-mer/sketch parameters (k=12, sketch=200): No contigs
- Ultra-permissible parameters (15 bp overlap, 0.75 identity, k=8, sketch=250): Successful assembly with 86.5% genome coverage

This demonstrates the critical importance of parameter tuning for OLC with noisy long reads, and the algorithm's innate ability to accommodate sequencing errors when properly configured.

5) Comparison Between Algorithms: Our experiments reveal distinct complementary strengths between the DBG and OLC approaches, with OLC demonstrating superior error tolerance:

a) Algorithm-Read Type Synergies:

- **DBG-Perfect Data Synergy**: The DBG algorithm excels with error-free data of both read types, producing near-perfect assemblies.
- **OLC-Error Tolerance Synergy**: The OLC algorithm demonstrates remarkable robustness to sequencing errors, particularly with long reads where it outperforms DBG by an enormous margin (86.5% vs. 18.2% genome coverage).

b) Error Sensitivity:

- **DBG Vulnerability**: The DBG approach shows extreme sensitivity to errors. Its reliance on exact k-mer matches means that errors can create massive graph complexity, branch points, and ultimately incorrect paths and contigs.
- **OLC Resilience**: The OLC approach shows impressive error tolerance when properly parameterized. It can accommodate sequence variation through flexible overlap criteria, and its use of weighted quality scores helps resolve discrepancies.

c) Assembly Error Profiles:

- **DBG Error Pattern**: Produces mostly mismatches (5,587 per 100 kbp for HiSeq) with error-containing data and many entirely falsely assembled regions (97% unaligned for ONT).
- **OLC Error Pattern**: Tends toward higher duplication (especially with ONT data) but maintains much lower mismatch rates (43.88 per 100 kbp for HiSeq) and better overall genome representation.

6) Key Insights: Our comprehensive analysis of MERS virus assembly reveals several important insights:

1) **Error Tolerance**: OLC demonstrates far superior error tolerance compared to DBG, which makes it the algorithm of choice for error-containing sequencing data, particularly for long reads.
2) **Algorithm Selection Criteria**:
   - For error-free data: Both algorithms perform well, with DBG producing more concise assemblies
   - For error-containing data: OLC is clearly superior, achieving better genome coverage with drastically lower error rates
   - For error-containing long reads: OLC is the only viable option, as DBG essentially fails with 82% of the genome missing
3) **Parameter Importance**: OLC requires careful parameter tuning for error-containing reads, but this investment pays off with dramatically better assemblies. The DBG approach offers fewer tuning options and cannot overcome the fundamental problem of error-corrupted k-mers.
4) **Quality vs. Contiguity Trade-offs**: OLC with error-containing data prioritizes accuracy over contiguity, while DBG might produce longer contigs but with extremely high error rates.
5) **Overassembly Risk**: DBG with error-containing reads creates severe overassembly issues, with most content being erroneous and unaligned to the reference.

7) Conclusions: The MERS virus assembly experiments provide compelling evidence that the OLC algorithm is substantially more robust to sequencing errors than the DBG approach. While DBG is effective for error-free data, it becomes increasingly unreliable as error rates rise.

OLC, with appropriate parameter tuning, can maintain acceptable assembly quality even with noisy data. This inherent error tolerance makes OLC particularly valuable for real-world sequencing data, where perfect reads are rarely available, and especially for error-prone long-read technologies like Oxford Nanopore.

These findings align with the historical development of assembly algorithms: DBG assemblers dominated in the era of short, high-accuracy reads, while OLC approaches have seen a resurgence with the advent of long, error-prone reads from third-generation sequencing technologies.

E. Comparison with Professional Tools

To contextualize the performance of our implementations, we compared them with SPAdes, a state-of-the-art de Bruijn graph-based assembler widely used in the bioinformatics community. SPAdes was run on the same MERS-CoV datasets (error-free and error-containing HiSeq and ONT reads) and evaluated using QUAST against the same reference genome.

1) SPAdes Assembly Results: Table IV presents a comparison of key metrics between SPAdes and our best performing implementations for each dataset type.

2) Performance Comparison for Error-free Data: For error-free data, our DBG implementation achieved performance nearly identical to SPAdes:

- **Genome Fraction**: Both achieved 97.9% (HiSeq) and 98.8% (ONT) coverage
- **Contiguity**: Both produced a single contig of identical length (29,482 bp for HiSeq, 29,748 bp for ONT)
- **Accuracy**: Both achieved perfect accuracy with zero mismatches or indels
- **Duplication Ratio**: Both maintained a perfect 1.000 ratio

This near-identical performance suggests that for clean, error-free data, our basic DBG implementation captures the core functionality of SPAdes' sophisticated algorithm. The simplicity of the MERS genome (limited repetitive content) combined with error-free reads creates an ideal scenario where the fundamental DBG approach is sufficient for optimal assembly.

Our OLC implementation, while achieving good genome coverage for error-free ONT data (97.6%), produced a more fragmented and duplicated assembly (6 contigs, 3.95x duplication) compared to SPAdes' single perfect contig.

3) Performance Gap for Error-containing Data: For error-containing data, SPAdes dramatically outperformed our implementations:

- **HiSeq Data**: While our DBG implementation achieved only 70.8% genome coverage with 8 contigs and extreme error rates (5,587 mismatches/100kbp), SPAdes maintained 97.9% coverage with a single, error-free contig.
- **ONT Data**: The gap was even more pronounced for error-prone ONT reads:
  - Our DBG implementation collapsed to just 18.2% genome coverage with severe overassembly
  - Our OLC implementation, despite extensive parameter tuning, achieved 86.5% coverage but with high duplication (2.86x) and error rates
  - SPAdes maintained 98.7% coverage with a single contig and modest error rates (43.7 mismatches/100kbp, 110.9 indels/100kbp)

4) Algorithmic Differences: The performance gap, particularly for error-containing data, highlights several advanced features in SPAdes that our implementations lack:

1) **Error Correction**: SPAdes likely incorporates sophisticated pre-assembly error correction, as evidenced by its ability to produce error-free contigs even from error-containing HiSeq data.

TABLE IV: Comparison of SPAdes with our implementations on MERS genome assembly

Metric	SPAdes				Our Best Implementation			
	Error-free HiSeq	**Error-free ONT**	**Error HiSeq**	**Error ONT**	**Error-free HiSeq**	**Error-free ONT**	**Error HiSeq**	**Error ONT**
# contigs	1	1	1	1	1 (DBG)	1 (DBG)	8 (DBG)	4 (OLC)
Largest contig	29,482 bp	29,748 bp	29,482 bp	29,751 bp	29,482 bp	29,748 bp	18,800 bp	20,731 bp
Total length	29,482 bp	29,748 bp	29,482 bp	29,751 bp	29,482 bp	29,748 bp	41,056 bp	74,432 bp
Genome fraction	97.9%	98.8%	97.9%	98.7%	97.9% (DBG)	98.8% (DBG)	70.8% (DBG)	86.5% (OLC)
Duplication ratio	1.000	1.000	1.000	1.000	1.000	1.000	1.008	2.855
Mismatches/100 kbp	0.00	0.00	0.00	43.70	0.00	0.00	5,587.09	298.39
Indels/100 kbp	0.00	0.00	0.00	110.94	0.00	0.00	669.89	1,149.19

2) **Read Pair Utilization**: SPAdes leverages paired-end information when available, which helps resolve repetitive regions and ambiguities.
3) **Multiscale Assembly**: SPAdes employs a multiscale approach that uses multiple k-mer sizes to build the de Bruijn graph, combining the advantages of small k-mers (error tolerance) and large k-mers (repeat resolution).
4) **Advanced Graph Processing**: SPAdes implements sophisticated graph cleaning algorithms to identify and resolve tips, bubbles, and chimeric connections that arise from sequencing errors.
5) **Coverage-Based Heuristics**: SPAdes likely uses coverage information to distinguish between genuine sequences and error-induced paths.
6) **Auto-Parameter Optimization**: Rather than requiring manual parameter tuning, SPAdes automatically optimizes parameters based on dataset characteristics.

5) Specific Advantages of SPAdes for Error-containing Data: The most remarkable aspect of SPAdes' performance is its error handling capability:

- **HiSeq Error Elimination**: SPAdes completely eliminated errors in HiSeq data, producing an error-free assembly identical to its error-free counterpart. This suggests powerful error correction mechanisms specific to Illumina error profiles.
- **ONT Error Management**: While SPAdes couldn't completely eliminate ONT errors, it managed them far more effectively than our implementations, maintaining assembly completeness without fragmentation or significant duplication.
- **Consistency Across Data Types**: SPAdes produced remarkably consistent results regardless of read type or error profile, indicating robust algorithms that adapt to different data characteristics.

6) Lessons and Potential Improvements: Comparing our implementations with SPAdes provides valuable insights for improving our assemblers:

1) **For DBG Implementation**:
   - Add preprocessing error correction, particularly for error-containing data
   - Implement graph cleaning operations (bubble popping, tip clipping)
   - Add coverage-based filtering to remove likely erroneous k-mers
   - Consider a multiscale approach using different k-mer sizes
2) **For OLC Implementation**:
   - Improve repeat detection and resolution to reduce duplication
   - Enhance the consensus generation algorithm to produce more accurate sequences
   - Implement adaptive parameter selection based on data characteristics
   - Add more sophisticated read error correction before overlap detection

7) Conclusions from Professional Tool Comparison: This comparison with SPAdes yields several important conclusions:

1) **Algorithm Fundamentals**: Our implementations capture the core principles of genome assembly algorithms, performing well on idealized data.
2) **Error Handling Gap**: The most significant difference between our implementations and professional tools lies in error handling, highlighting the importance of error correction and graph processing in real-world assembly.
3) **Implementation Sophistication**: Professional assemblers like SPAdes represent years of algorithm refinement and optimization beyond the basic algorithms we implemented.
4) **Real-world Applicability**: For practical genome assembly tasks with error-containing data, professional tools like SPAdes remain the preferred choice due to their superior error handling and consistency.
5) **Educational Value**: Despite performance gaps, our implementations provide valuable insights into the fundamental algorithms underlying genome assembly and the challenges involved in handling real sequencing data.

The comparison with SPAdes serves as an excellent benchmark and provides a roadmap for future improvements to our implementations. It also highlights the remarkable advances in genome assembly algorithms, which have evolved from basic theoretical frameworks to sophisticated tools capable of producing high-quality assemblies from imperfect real-world data.

IV. Task 2: De Novo Assembly of Scincus mitranus Genome

The second part of this study focused on the de novo assembly of the Scincus mitranus (sandfish lizard) genome

using state-of-the-art assembly workflows. This represents a significant step up in complexity from the synthetic datasets in Task 1, involving a vertebrate genome with real sequencing data from multiple technologies.

A. Assembly Approach

For this task, we selected Verkko as our assembly tool, a graph-based assembler specifically designed to leverage the complementary strengths of PacBio HiFi (high accuracy) and Oxford Nanopore (long reads) technologies.

1) Data Preparation: The assembly utilized two primary data sources:

- **PacBio HiFi**: High-accuracy long reads from liver tissue (lizard_liver_seq.fastq.gz)
- **Oxford Nanopore**: Ultra-long reads (lizard_ont.fastq.gz)

2) Computational Resources: The assembly was performed on the Ibex High-Performance Computing (HPC) cluster with the following resources:

- 32 CPU cores
- 256 GB RAM
- 72-hour time allocation (though the runs completed faster)

B. Assembly Process and Challenges

The assembly process encountered several challenges that provided valuable insights into real-world genome assembly projects:

1) **Version Compatibility**: An initial assembly attempt failed after 7 hours due to loading the wrong Verkko version by default. This highlights the importance of version control and compatibility in bioinformatics workflows.
2) **Multiple Assembly Strategies**: We conducted two parallel assembly approaches:
   - **Primary Assembly**: Verkko v2.2.1 with ONT and HiFi reads
   - **Backup Assembly**: Verkko v1.4.1 with ONT, HiFi, and Hi-C reads
3) **Runtime Differences**: The v2.2.1 assembly completed in approximately 18 hours, while the v1.4.1 assembly took about 25 hours, demonstrating significant performance improvements in the newer version.

C. Assembly Results and Comprehensive Evaluation

The assembly was generated using Verkko v2.2.1, combining PacBio HiFi and Oxford Nanopore reads. We then conducted a comprehensive evaluation using multiple complementary approaches to assess assembly quality.

1) QUAST Assembly Metrics: QUAST analysis revealed excellent contiguity and completeness:

2) BUSCO Assessment Challenges: We attempted to assess gene completeness using BUSCO (Benchmarking Universal Single-Copy Orthologs), but encountered significant computational challenges:

- Multiple attempts failed due to Out of Memory (OOM) errors despite allocating substantial resources:
  - 32 threads with 128 GB RAM: Failed
  - 32 threads with 256 GB RAM: Failed
  - 8 cores with 256 GB RAM: Failed after approximately 7 hours
- A modified "chunked" approach was also attempted without success

These failures likely reflect the exceptional size and complexity of the sandfish genome, potentially including expanded gene families that require computational resources beyond those typically needed for reptile genomes.

3) Merqury K-mer Analysis: Merqury provided valuable insights into assembly completeness and accuracy through k-mer analysis:

TABLE VI: Merqury k-mer statistics (k=21) for Scincus mitranus genome assembly

Metric	Value
Unique k-mers	1,061,469,078
Distinct k-mers	3,038,087,090
Present k-mers	30,047,684,002
Missing k-mers	4,395,008,424,014
QV Score	42
Estimated error rate	0.0000631 errors per base

The QV (Quality Value) score of 42 indicates exceptional base-level accuracy, roughly equivalent to 99.994% accuracy. This is considered excellent for a de novo assembly of a complex vertebrate genome.

4) Inspector Assembly Assessment: Inspector provided a detailed structural and small-scale error assessment:

TABLE VII: Inspector assembly error assessment for Scincus mitranus genome

Metric	Value
Structural Errors	
Total structural errors	47
Expansion errors	21
Collapse errors	26
Haplotype switch errors	0
Inversion errors	0
Small-Scale Errors	
Small-scale errors per Mbp	35.01
Total small-scale errors	122,262
Base substitutions	102,782
Small-scale expansions	10,670
Small-scale collapses	8,810
QV score	44.13
Read Alignment Statistics	
Overall mapping rate	99.99%
Split-read rate	0.74%
Average depth	8.61
Mapping rate in large contigs	78.14%
Split-read rate in large contigs	0.59%
Depth in large contigs	8.73

The Inspector assessment reveals several remarkable features of the assembly:

1) **Minimal Structural Errors**: Only 47 structural errors across the entire 3.49 Gb genome, with a balanced distribution between expansions and collapses

TABLE V: QUAST evaluation metrics for Scincus mitranus genome assembly

Metric	Value
Total assembly size	3,491,931,429 bp (3.49 Gb)
Number of contigs (3000 bp)	3,132
Largest contig	12,917,798 bp (12.92 Mb)
GC content	45.38%
N50	2,044,959 bp (2.04 Mb)
N90	546,269 bp
L50	493 contigs
L90	1,752 contigs
N's per 100 kbp	0.00

2) **No Complex Structural Issues**: Zero haplotype switches or inversions, which are typically the most challenging errors to resolve
3) **Excellent Base-Level Accuracy**: Inspector's QV score of 44.13 (even higher than Merqury's 42) indicates approximately 99.996% base-level accuracy
4) **Comprehensive Genome Representation**: The near-perfect mapping rate (99.99%) confirms that the assembly captures virtually all of the sequenced DNA
5) **Low Split-Read Rate**: The minimal split-read rate (0.74%) indicates few misassemblies or breakpoints

D. Integrated Assembly Quality Assessment

Synthesizing the results from QUAST, Merqury, and Inspector provides a comprehensive assessment of the Scincus mitranus genome assembly:

1) Contiguity and Completeness:

- The assembly demonstrates exceptional contiguity for a non-model organism, with an N50 of 2.04 Mb
- The largest contig spans nearly 13 Mb, and over 1,155 contigs exceed 1 Mb in length
- The complete absence of gaps (0.00 N's per 100 kbp) confirms a fully resolved assembly with no ambiguous regions
- The near-perfect read mapping rate (99.99%) verifies that the assembly comprehensively represents the sequenced DNA

2) Accuracy and Error Profile:

- Both Merqury (QV: 42) and Inspector (QV: 44.13) confirm exceptional base-level accuracy
- The minimal number of structural errors (47) across 3.49 Gb is remarkably low for a first assembly
- The small-scale error rate (35.01 per Mbp) is well within acceptable ranges for a high-quality reference
- Most errors (102,782 out of 122,262) are simple base substitutions, which are the easiest to correct in subsequent polishing steps

3) Assembly Robustness:

- The balanced distribution of expansion (21) and collapse (26) errors suggests no systematic bias in the assembly process
- The absence of haplotype switches and inversions indicates excellent resolution of complex genomic regions
- The high percentage of reads mapping to large contigs (78.14%) confirms that most of the genome resides in well-assembled regions

E. Comparison with Other Reptile Genomes

To contextualize our assembly, we can compare it with other reptile genome assemblies:

This comparison highlights several notable features of our assembly:

- The Scincus mitranus genome is significantly larger than most other sequenced reptile genomes, suggesting potential genomic expansions
- Despite its size, the contiguity metrics (N50 of 2.04 Mb) are competitive with the best reptile assemblies
- The QV score of 42-44 indicates exceptional accuracy, though QV metrics are not commonly reported for older assemblies

F. Biological Significance

The high-quality genome assembly of Scincus mitranus provides a valuable resource for studying:

- **Desert Adaptations**: Sandfish lizards are known for their unique "swimming" behavior in sand and special adaptations for desert environments.
- **Reptile Evolution**: This genome contributes to our understanding of squamate reptile evolution and diversity.
- **Comparative Genomics**: The assembly enables comparisons with other reptiles to identify genomic innovations specific to this desert specialist.
- **Genome Size Evolution**: The unusually large genome size (3.49 Gb) compared to other reptiles warrants investigation into potential genomic expansions or repetitive element proliferation.

G. Recommendations for Further Improvement

Despite the already excellent quality of this assembly, several steps could further enhance its utility:

1) **Targeted Error Correction**: The small number of structural errors (47) could be manually inspected and corrected, potentially elevating the assembly to reference quality.
2) **Additional Polishing**: While base-level accuracy is already exceptional (QV ¿ 40), additional polishing could address the identified substitution errors.

TABLE VIII: Comparison with other reptile genome assemblies

Species	Genome Size	N50	Contigs	QV	Reference
Scincus mitranus (This study)	3.49 Gb	2.04 Mb	3,132	42-44	-
Anolis carolinensis	1.78 Gb	4.03 Mb	2,213	N/A	Alföldi et al., 2011
Python bivittatus	1.44 Gb	207.5 kb	39,113	N/A	Castoe et al., 2013
Pogona vitticeps	1.82 Gb	2.29 Mb	2,092	N/A	Georges et al., 2015
Salvator merianae	2.51 Gb	2.12 Mb	2,823	N/A	Fallon et al., 2021

3) **Hi-C Scaffolding**: Incorporating Hi-C data would enable chromosome-level organization of the contigs, providing valuable synteny information.
4) **Alternate BUSCO Strategy**: Given the computational challenges encountered, alternative approaches to gene completeness assessment could be explored, such as using lineage-specific BUSCO datasets, reducing the search space, or using specialized high-memory computing resources.
5) **Annotation Pipeline**: Gene prediction and functional annotation would transform this assembly into a comprehensive genomic resource.

H. Conclusions from the Sandfish Genome Project

The successful generation of a high-quality genome assembly for Scincus mitranus represents a significant achievement in reptile genomics and highlights several important principles in modern genome assembly:

1) **Hybrid Assembly Power**: The combination of PacBio HiFi and Oxford Nanopore technologies through Verkko has produced an assembly of exceptional quality, demonstrating the synergistic benefits of integrating complementary sequencing approaches.
2) **Quality Beyond Contiguity**: While contiguity metrics like N50 have traditionally dominated assembly quality discussions, our comprehensive evaluation reveals the equally important dimensions of accuracy (QV scores), structural integrity (minimal errors), and completeness (mapping rates).
3) **Computational Challenges**: The BUSCO analysis failures highlight the computational demands of working with large, complex genomes. As assembly contiguity improves, computational methods for downstream analysis must similarly advance to handle increasingly complete genomic representations.
4) **Reptile Genomic Diversity**: The unusually large genome size of Scincus mitranus compared to other sequenced reptiles underscores the genomic diversity within Reptilia and suggests potentially interesting evolutionary phenomena that warrant further investigation.
5) **Technological Maturation**: The quality of this assembly, achieved without reference-guided approaches, demonstrates the maturation of de novo assembly technologies to a point where high-quality reference genomes can be produced for non-model organisms.

This assembly provides a solid foundation for future studies on the biology, evolution, and adaptation of this fascinating desert specialist and contributes a valuable resource to the growing repository of reptile genomes. The exceptional quality metrics suggest it could serve as a reference-quality genome with only minimal additional refinement, potentially benefiting conservation efforts and comparative genomic studies across reptiles.

REFERENCES

[1] Compeau, P.E., Pevzner, P.A. and Tesler, G., 2011. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, 29(11), pp.987–991.
[2] Li, H., 2016. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14), pp.2103–2110.
[3] Zerbino, D.R. and Birney, E., 2008. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, 18(5), pp.821–829.
[4] Bankevich, A., Nurk, S., Antipov, D., Gurevich, A.A., Dvorkin, M., Kulikov, A.S. et al., 2012. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5), pp.455–477.
[5] Koren, S., Walenz, B.P., Berlin, K., Miller, J.R., Bergman, N.H. and Phillippy, A.M., 2017. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5), pp.722–736.
[6] Gurevich, A., Saveliev, V., Vyahhi, N. and Tesler, G., 2013. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8), pp.1072–1075.
[7] Wick, R.R., Schultz, M.B., Zobel, J. and Holt, K.E., 2015. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, 31(20), pp.3350–3352.
[8] Rhie, A., Walenz, B.P., Koren, S. and Phillippy, A.M., 2020. Merqury: reference-free quality, completeness, and phasing assessment for genome assemblies. *Genome biology*, 21(1), pp.1–27.
[9] Garg, S., Aach, J., Li, H., Sebenius, I., Durbin, R. and Church, G., 2020. A haplotype-aware de novo assembly of related individuals using pedigree sequence graph. *Bioinformatics*, 36(8), pp.2385–2392.
[10] Simão, F.A., Waterhouse, R.M., Ioannidis, P., Kriventseva, E.V. and Zdobnov, E.M., 2015. BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19), pp.3210–3212.
[11] Garrison, E. and Marth, G., 2012. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907*.
[12] Nurk, S., Koren, S., Rhie, A., Rautiainen, M., Bzikadze, A.V., Mikheenko, A. et al., 2022. The complete sequence of a human genome. *Science*, 376(6588), pp.44–53.
[13] Chin, C.S., Alexander, D.H., Marks, P., Klammer, A.A., Drake, J., Heiner, C. et al., 2013. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature methods*, 10(6), pp.563–569.
[14] Lieberman-Aiden, E., Van Berkum, N.L., Williams, L., Imakaev, M., Ragoczy, T., Telling, A. et al., 2009. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950), pp.289–293.
[15] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J., 1990. Basic local alignment search tool. *Journal of molecular biology*, 215(3), pp.403–410.
[16] Li, H., 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18), pp.3094–3100.
[17] Rautiainen, M., Mäkinen, V. and Marschall, T., 2019. Bit-parallel sequence-to-graph alignment. *Bioinformatics*, 35(19), pp.3599–3607.
[18] Mikheenko, A., Prjibelski, A., Saveliev, V., Antipov, D. and Gurevich, A., 2018. Versatile genome assembly evaluation with QUAST-LG. *Bioinformatics*, 34(13), pp.i142–i150.

[19] Broder, A.Z., 1997. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences* (pp. 21–29).

[20] Ondov, B.D., Treangen, T.J., Melsted, P., Mallonee, A.B., Bergman, N.H., Koren, S. and Phillippy, A.M., 2016. Mash: fast genome and metagenome distance estimation using MinHash. *Genome biology*, 17(1), pp.1–14.

[21] Marçais, G. and Kingsford, C., 2011. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6), pp.764–770.