Assignment 2 for CS249: Assembly

Zeyad Aljaali 179307

May 11, 2025

1 Task 1.1: De Bruijn Graph (DBG) Assembly

https://github.com/ZeyadAl/CS249-Assignment2

2 Task 1.2: Overlap-Layout-Consensus (OLC) Assembly

https://github.com/ZeyadAl/CS249-Assignment2

3 Task 1.3: Applications of assembly algorithms

1. Use your De Bruijn Graph implementation to construct an assembly graph for reads b.fastq with k = 40. Either export the graph in GFA format (e.g., using gfatools), or directly write GFA format as part of your assembly. Visualize the graph using Bandage (https://rrwick.github.io/Bandage/). Describe and explain what you see. How can this help you to improve the assembly?

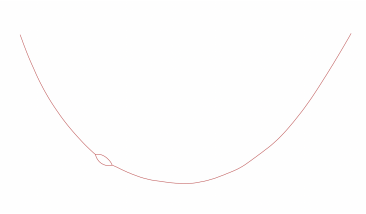

Figure 1: DBG for reads_b.fastq with `k=40`.

As we can see, the path to follow to assemble the genome is clear but for one bubble that we have. This means that the assembly process should not be too difficult, however we expect the algorithm to output 4 contigs. We can improve the assembly by increasing changing k, or by looking at the degrees of each side of the bubble.

2. Apply your DBG implementation to reads_r.fastq with `k=35` and `k=45`. Then evaluate it with QUAST against reference_r.fasta. Include the QUAST evaluation metrics, a visualization of the assembly graphs using Bandage, a brief description of the differences between the two assemblies, and an explanation for the observed difference (if any).

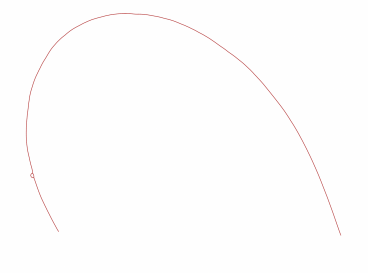

Figure 2: DBG for reads_r.fastq with `k=35`.

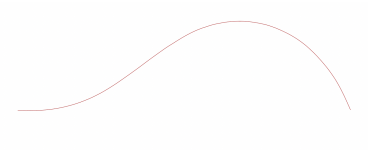

Figure 3: DBG for reads_r.fastq with `k=45`.

We notice that when `k=35` we have a small bubble and we get 4 contigs, but when `k=45` there are no bubbles and we get only 1 contig.

Metric	**k=35**	**k=45**
Total assembly length (bp)	1136	1040
Number of contigs	4	1
GC content (%)	50.79	51.25
Genome fraction (%)	100.00	100.00
Duplication ratio	1.008	1.000
Largest contig (bp)	914	1040
N50 (bp)	914	1040
N90 (bp)	134	1040
L50 (contigs)	1	1
Misassemblies	0	0
Mismatches per 100 kbp	0.00	0.00
Indels per 100 kbp	0.00	0.00

Table 1: QUAST assembly statistics for $k = 35$ vs. $k = 45$.

For these reads and target genome, k = 45 provides a perfect, single-contig reconstruction with no misassemblies, whereas k = 35 yields a fragmented assembly with minor redundancy and a small graph bubble. The reason a larger k resolves the bubble is that the repeats are not longer than the k-mer anymore. This allows the algorithm to find the correct assembly unambiguously.

3. For a more realistic analysis, assemble the MERS virus reads using your two algorithms. Use both algorithms you implemented for "hiseq" reads (separately for reads with and without errors) and "ont" reads (also separately for reads with and without errors). Evaluate all assemblies using QUAST and compare with the MERS reference genome. Include in your report the QUAST evaluation metrics for all assemblies, comparison between error-free and error-containing assemblies, analysis of how each algorithm handles the different reads and errors.

Metric	**ONT (no errors)**	**HiSeq (no errors)**	**ONT (with errors)**	**HiSeq (with errors)**
Total assembly length (bp)	29748	29482	1496239	87740
Number of contigs	1	1	23729	1301
GC content (%)	41.27	41.26	40.82	41.25
Genome fraction (%)	98.768	97.885	90.617	90.816
Duplication ratio	1.000	1.000	2.569	1.279
Largest contig (bp)	29748	29482	2939	597
N50 (bp)	29748	29482	69	69
N90 (bp)	29748	29482	35	41
L50 (contigs)	1	1	5997	442
# Misassemblies	0	0	0	0
# Mismatches per 100 kbp	0.00	0.00	707.38	42.87
# Indels per 100 kbp	0.00	0.00	3140.42	0.00

Table 2: QUAST metrics for DBG assemblies at $k = 35$.

Metric	**ONT (no errors)**	**HiSeq (no errors)**	**ONT (with errors)**	**HiSeq (with errors)**
Total assembly length (bp)	29431	30215	20439	111742
Number of contigs	1	1	1	254
GC content (%)	41.17	41.30	40.58	41.50
Genome fraction (%)	97.716	97.892	67.638	84.508
Duplication ratio	1.000	1.000	1.003	2.202
Largest contig (bp)	29431	30215	20439	19677
N50 (bp)	29431	30215	20439	3141
N90 (bp)	29431	30215	20439	630
L50 (contigs)	1	1	1	6
# Misassemblies	0	0	0	3
# Mismatches per 100 kbp	0.00	0.00	230.01	599.45
# Indels per 100 kbp	0.00	0.00	734.07	271.18

Table 3: QUAST metrics for OLC assembly. (all stats are based on contigs at least 500 bp)

We notice overall that as we would expect, reads with no errors perform significantly better. In regards to DBG, we notice that it does very well on error-free reads, but it does significantly worse on error-containg reads. The reason goes back to using an appropriate k. When we don't do error correction this leads to us having way more k-mers that we need to collapse together somehow, and in this implementation, we don't deal particularly

well with that. We need to incorporate ways to deal with bubbles. OLC on the other hand, again performs well with error-free reads, but not as terrible on ONT reads, the reason being in the consensus step we can resolve regions easily by 'voting'.

How Each Algorithm Handles Different Reads and Errors

1. **DBG (De Bruijn Graph) Algorithm:**
    - **Error-free Reads:**
        - **Contiguity & Assembly Quality:** The DBG algorithm performs well with error-free reads, producing a single contig for both ONT and HiSeq data. The genome fraction remains high (close to 98%).
        - **Efficiency:** In this case, the algorithm shows high efficiency, as there is minimal fragmentation of the genome, and it handles the data efficiently.
    - **Error-free Reads:**
        - **Contiguity:** When faced with error-containing reads, the DBG algorithm performs poorly. It produces a fragmented assembly, leading to tens of thousands of smaller contigs. This happens because errors like mismatches and indels can break the contigs into smaller fragments, making it harder to assemble longer, continuous sequences.
        - **Genome Fraction & Duplication:** The genome fraction decreases significantly due to errors. The duplication ratio increases as well, indicating some repetitive regions are not resolved properly.
        - **Mismatch Handling:** Mismatches are more frequent in error-containing reads, with DBG showing mismatches in the order of 700/100 kbp for ONT errors. This results in a fragmented assembly and a lower-quality output.
2. **OLC (Overlap-Layout-Consensus) Algorithm:**
    - **Error-free Reads:**
    - **Contiguity and Assembly Quality:** OLC performs well with error-free reads, similar to DBG. It produces a single contig with good assembly quality. The genome fraction is very high, and the assembly is largely contiguous, making it suitable for high-quality reads.
    - **Handling Repetitive Regions:** Handling Repetitive Regions: The OLC algorithm handles repetitive regions well, as it employs overlap-based techniques that can capture even complex regions effectively without much fragmentation.
    - **Error-free Reads:**

- **Contiguity:** In error-containing reads, OLC does better than DBG. While still producing fragmented assemblies, it maintains a higher genome fraction (around 90%) and fewer contigs. This suggests that OLC is more robust in handling errors compared to DBG.
- **Missassimblies and Duplication:** While the duplication ratio remains relatively lower than DBG, OLC still struggles with misassemblies in error-containing reads. This can be seen from the increased number of misassemblies (e.g., 3 misassemblies in OLC for HiSeq with errors).
- **Mismatch Handling:** OLC handles mismatches and indels better than DBG, but still faces difficulties when dealing with errors. The number of mismatches and indels rises with error-containing reads, but the increase is less severe compared to DBG. For example, OLC shows fewer mismatches (around 700/100 kbp for ONT errors).

To summarize, OLC is better at handling error-containing reads due to its overlap-based approach, while DBG tends to struggle more as errors break contigs and lead to fragmentation. Both algorithms perform well with error-free reads, but OLC shows a clear advantage in handling errors without severely compromising the quality of the assembly.

4. Repeat the same assembly and analysis using either the SPAdes or Canu assembler. Compare the assembly results with the results obtained using your own algorithms. Explain any differences you observe.

Metric	Hybrid (no errors)	HiSeq (no errors)	ONT (no errors)	Hybrid (with errors)	ONT (with errors)	HiSeq (with errors)
Total assembly length (bp)	29,482	29,482	29,748	29482	29,482	29,754
Number of contigs	1	1	1	1	1	1
GC content (%)	41.26	41.26	41.27	41.26	41.26	41.27
Genome fraction (%)	97.885	97.885	98.768	97.885	97.885	98.738
Duplication ratio	1	1	1	1	1	1
Largest contig (bp)	29482	29482	29748	29482	29482	29754
N50 (bp)	29482	29482	29748	29482	29482	29754
N90 (bp)	29482	29482	29748	29482	29482	29754
L50 (contigs)	1	1	1	1	1	1
Misassemblies	0	0	0	0	0	0
Mismatches per 100 kbp	0	0	0	0	0	26.9
Indels per 100 kbp	0	0	0	0	0	73.96

Table 4: QUAST metrics for SPAdes assemblies.

Comparison with custom algorithms

* **Contiguity & completeness:** SPAdes produced a single contig for all conditions—even in the presence of sequencing errors, whereas for example our DBG at $k = 35$ (for ONT+errors) yielded tens of thousands of small contigs. In general our DBG/OLC methods both fragmented error-containing or got a low genome fraction.
* **Error tolerance:** While our own implementations had zero indels and mismatches on error-free data, they performed poorly

on error-containing reads (mismatches around 700 per 100 kbp for ONT errors in OLC. around 42 per 100 kbp for HiSeq errors). SPAdes' built-in error correction stages (BayesHammer, mismatches correction) drastically reduced both mismatches and indels, resulting in near-perfect assemblies under all conditions.

* **Repeat resolution:** SPAdes' iterative multi-k strategy allows it to resolve repeats shorter than its largest k, effectively collapsing bubbles without manual tuning. In contrast, our single-k DBG needed manual k adjustment ($k = 45$ vs. $k = 35$) to eliminate bubbles, and our OLC lacked robust repeat-resolution heuristics.
* **Runtime and resource usage:** Although SPAdes is more resource-intensive (due to multiple assemblies and correction passes), it remains practical and delivers consistently high-quality outputs. Our lightweight implementations run faster on small datasets but at the cost of error handling and repeat resolution.
* **Conclusion:** SPAdes outperforms the custom DBG/OLC pipelines on error-containing data by integrating advanced error correction and multi-k graph assembly, yielding single-contig, high-accuracy reconstructions without manual parameter tuning.

4 Task 2.1: Genome assembly

We assembled the genome using Hifiasm. The results can be found at:

`/ibex/user/aljaalza/Data-science-onboarding/launch_jupyter_server/249/2/asm/main.fa`

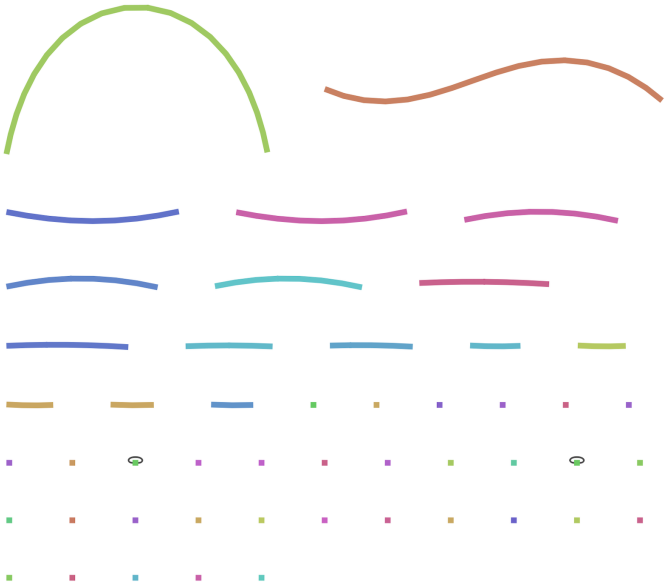

Figure 4: Visualization of hifiasm contigs using Bandage.

5 Task 2.2: Assembly evaluation

1. Provide basic metrics, using QUAST
2. Report gene completeness, using BUSCO or asmgene. We ran the command:

   ```
   busco -i main.fasta -o busco_auto --mode genome --auto-lineage --cpu 24
   ```

   This tests on two datasets: generic: eukaryota , and specific: squamata

Metric	Assembly using Hifiasm
Total assembly length (bp)	1,808,305,840
Number of contigs	49
GC content (%)	45.46
Genome fraction (%)	(No reference)
Duplication ratio	(No reference)
Largest contig (bp)	341,750,303
N50 (bp)	138,451,086
N90 (bp)	40,553,089
L50 (contigs)	4
Misassemblies	(No reference)
Mismatches per 100 kbp	(No reference)
Indels per 100 kbp	(No reference)

Table 5: QUAST metrics for Hifiasm assembly using all the reads based on contigs at least 500 bp long.

Metric	squamata (specific)	eukaryota (generic)
Complete percentage (%)	97.4	99.2
Complete BUSCOs	11000	128
Single copy percentage	97.2	98.4
Single copy BUSCOs	10975	127
Multi copy percentage	0.2	0.8
Multi copy BUSCOs	25	1
Fragmented percentage	1.4	0.8
Fragmented BUSCOs	162	1
Missing percentage	1.2	0.0
Missing BUSCOs	132	0
n_markers	11294	129
avg_identity	0.87	0.76
domain	eukaryota	eukaryota
internal_stop_codon_count	586	4
internal_stop_codon_percent	5.3%	3.1%

Table 6: BUSCO metrics for hifiasm assembly.

3. Report the k-mer distribution and QV score, using Merqury
   To report the k-mer distribution and QV score, we use Merqury. The best k-value is determined using the following command:

   ```
   sh $MERQURY/best_k.sh <genome_size> [tolerable_collision_rate=0.001]
   ```

This yields a best k-value of 20.3582. So we use $\mathbf{k = 21}$. We find that the QV score

Metric	All reads using Merqury	Hi-C reads using Merqury	All reads using Yak	All reads using Yak
QV score	75.9406	61.0953	61.868	61.868

Table 7: QUAST metrics for Hifiasm assembly using all the reads.

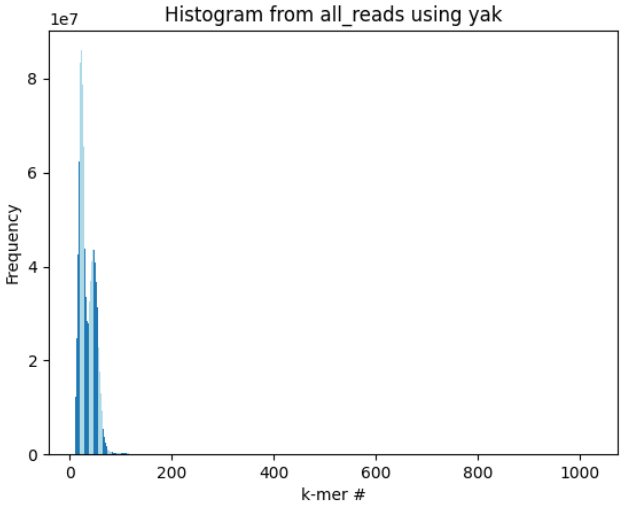

Figure 5: k-mer distribution using `yak`.

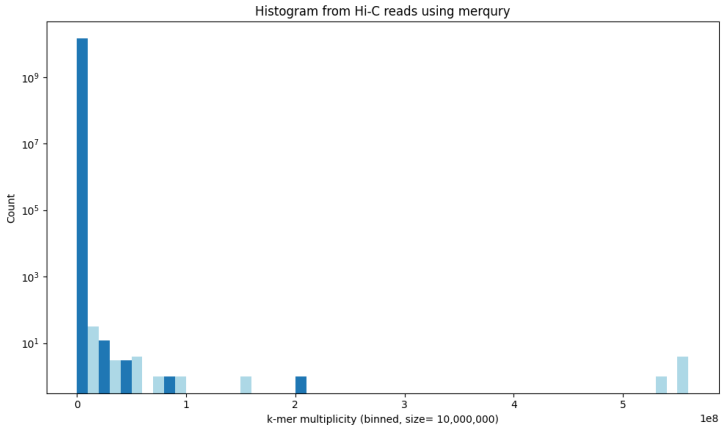

Figure 6: k-mer distribution using `Merqury`.

4. identify mis-assemblies, using Flagger or Inspector

Metric	Value
Structural error	2
Expansion	2
Collapse	0
Haplotype switch	0
Inversion	0
Small-scale assembly error (per Mbp)	2833.98907786528
Total small-scale assembly error	5124719
Base substitution	4742045
Small-scale expansion	245278
Small-scale collapse	137396

Table 8: Inspector mis-assembly and small-scale error metrics

5. Provide a basic report for each evaluation step, and explain what the measures mean with respect to the assembly. Can you identify ways to improve the assembly based on the evaluation?

   Basic metrics (QUAST): The total assembly length (1.81 Gb) is in line with the expected genome size for a Squamata member, indicating near-complete representation of the genome. A low contig count (49) together with a high N50 (138 Mb) and low L50 (4) testify to excellent contiguity. The GC content (45.46 %) matches published values for related lizard species, suggesting accurate base composition.
   Take-home: the assembly is highly contiguous and of the expected size, but without a reference we cannot directly assess genome fraction or duplication ratio.

   Gene completeness (BUSCO): On the Squamata lineage, BUSCO finds 97.4 % complete genes, with only 1.4 % fragmented and 1.2 % missing. Against the broader Eukaryota set, completeness is 99.2 %, with 0.8 % fragmented and no missing markers. The low duplication rate indicates minimal collapsed regions.
   Take-home: gene space is almost fully recovered; remaining fragmentation and frameshift errors can be polished further.

   k-mer spectrum & QV (Merqury): With $k = 21$, Merqury reports a consensus quality (QV) of 75.94 for the full read set, corresponding to an error rate of $\approx 2.6 \times 10^{-8}$. The Hi-C reads give a somewhat lower QV (61.10), reflecting their higher base-error profile.
   Take-home: base accuracy is excellent; residual errors are at the

small-variant level.

Mis-assemblies (Inspector): Only 2 structural errors (expansions) and zero inversions or collapses indicates very few large-scale mis-joins. However, there are around 2.8 k small errors per Mbp (4.7 M substitutions, 245 k small indels, 137 k small collapses), pointing to base-level inconsistencies or local mis-alignments.
Take-home: the scaffold graph is largely correct, but fine-scale polishing is needed.

Improvement Strategies

* **Polishing:** Run a round of consensus polishing using short reads indels, reducing the small-scale error rate.
* **Gap closure:** Use local assembly around BUSCO fragments to join any fragmented gene models.
* **Hi-C scaffolding:** We can use a Hi-C scaffolder to order and orient contigs into chromosome-level scaffolds, further reducing misassemblies and improving L50.
* **Coverage balancing:** If any genomic regions are under-represented, consider supplementing with additional HiFi or ONT coverage to even out read depth.

Overall, the Hifiasm assembly is highly contiguous, complete, and accurate. Polishing and scaffolding will elevate it to a reference-quality draft.

6 Challenges

There were two main challenges

* Understanding and coding the algorithms. I dealt with this by watching Youtube videos (by Ben Langmead) and looking up slides online to make sure I understood it correctly. Specifically, lectures 14 and 15 of https://rob-p.github.io/CSE549F16/lectures/.
* Running the tools. I had to read the documentations, and make sure I requested the appropriate tools on ibex.

7 Collaboration

I have used ChatGPT o4-mini-high. I used it for formatting the LaTeXtables and some of the writing in the report, as well as in coding assistance in functions and documenting the code. I validated the correctness by manually inspecting the results and logic, as well as testing on a small test cases.