Bioinformatics Algorithms
Assignment 2: Genome Assembly and Evaluation

April 10, 2025

1 Introduction

This assignment focuses on genome assembly algorithms and their evaluation. Genome assembly is the process of reconstructing a complete genome sequence from fragmented DNA sequence "reads". You will implement two fundamental assembly approaches, apply them to both synthetic and real datasets, and evaluate the results using quantitative measures.

1.1 Datasets

All data is available at `https://bio2vec.cbrc.kaust.edu.sa/data/mowl/cs249_hw2.tar.gz`.

1.1.1 Synthetic Dataset 1

We provide a small entirely synthetic dataset you can use for testing your algorithms:

- `reference_r.fasta`: Reference sequence 1
- `reads_r.fastq`: First set of simulated reads (for reference 1)
- `reference_b.fasta`: Reference sequence 2
- `reads_b.fastq`: Second set of simulated reads (for reference 2)

1.1.2 Synthetic Dataset 2

This dataset contains simulated reads from the Middle East respiratory syndrome-related coronavirus (MERS-CoV) genome:

- Reference genome: `GCF_000901155.1_ViralProj183710_genomic.fna`
- Error-free Illumina HiSeq simulated reads (approximately 50x coverage):
  - `no_error_reads_hiseq_5k.fastq`
- Error-free Oxford Nanopore Technologies (ONT) simulated reads:
  - `no_error_ont_hq_50x.fastq`
- Illumina HiSeq simulated reads with errors:
  - `reads_hiseq_5k.fastq`
- ONT simulated reads with errors:
  - `ont_hq_50x.fastq`

1.1.3 Real Dataset

For the real dataset, you will use sequencing data from *Scincus mitranus* (a sandfish lizard species), available from NCBI SRA under study accession `SRP563043`.

- Hi-C Illumina NovaSeq 6000 paired-end reads (`SRR32302809`)
- Oxford Nanopore PromethION reads (`SRR32302812`)
- PacBio Sequel II reads (`SRR32302813`)
- PacBio Revio reads (`SRR32302814`)

For genome annotation, you can also use this data:

- Lizard eye RNA-Seq PacBio Sequel IIe reads (`SRR32302810`)
- Lizard liver RNA-Seq PacBio Sequel IIe reads (`SRR32302811`)

1.2 Software tools and resources

- **QUAST** (`http://quast.sourceforge.net/quast`): Assembly evaluation tool
- **Bandage** (`https://rrwick.github.io/Bandage/`): Assembly graph visualization

- **SPAdes** (https://github.com/ablab/spades): DBG-based assembler for Illumina reads
- **Canu** (https://github.com/marbl/canu): OLC-based assembler for long reads
- **SRA Toolkit** (https://github.com/ncbi/sra-tools): For downloading SRA data
- **seqtk** (https://github.com/lh3/seqtk): For subsampling FASTQ files
- **GFA Format Specification**: https://github.com/GFA-spec/GFA-spec

Additionally, most tools you need here are pre-installed on the Ibex cluster (use `module avail`), and necessary resources are also installed on Ibex (folder: `/ibex/reference/`, in particular `/ibex/reference/KSL/`).

2 (60 pts) Task 1: Basic genome assembly algorithms

In this task, you will implement two fundamental genome assembly algorithms. Code should be documented and include a README with usage instructions. Provide a basic command-line interface for running the assembly algorithms and example output in your README.

For all assemblies, calculate and report the following metrics (some of these require only the assembly, others require both the assembly and a reference against which it is compared):

Sequence length Total assembly length

Number of contigs Total number of contigs

GC content (%) GC percentage

Genome fraction (%) % of reference covered by assembly

Duplication ratio Average number of times a reference base is covered

Largest contig Length of the largest contig

N50 Length where 50% of assembly is in contigs of this size or larger

N90 Length where 90% of assembly is in contigs of this size or larger

L50 Number of contigs to reach 50% of total assembly length

Misassemblies Number of positions with breakpoints relative to reference

Mismatches per 100 kbp Number of mismatches per 100,000 bases

Indels per 100 kbp Number of insertions/deletions per 100,000 bases

2.1 Task 1.1: De Bruijn Graph (DBG) Assembly

Implement a De Bruijn Graph assembly algorithm that:

- Takes FASTQ files as input
- Constructs a de Bruijn graph from k-mers (with user-defined k)
- Identifies contigs by finding Eulerian paths
- Outputs contigs as a FASTA file

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

Implement an OLC assembly algorithm that:

- Takes FASTQ files as input
- Computes all-vs-all read overlaps and constructs an overlap graph, using the minimum overlap length n as a parameter
- Identifies non-branching paths in the graph
- Generates a layout of reads
- Computes a consensus sequence for each contig
- Outputs contigs as a FASTA file

2.3 Task 1.3: Applications of assembly algorithms

1. (12 pts) 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?

2. (12 pts) Apply your DBG implementation to `reads_r.fastq` with k ≤ 40 (e.g., $k = 35$). Evaluate the assembly using QUAST against the reference `reference_r.fasta`. Repeat the assembly with $k > 40$ (e.g., $k = 45$). Compare the two assemblies and explain the differences. Your report should 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).

3. (24 pts, optionally 6 pts) 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.

4. (12 pts) Repeat the same assembly and analysis using either the SPAdes (https://github.com/ablab/spades) or Canu (https://github.com/marbl/canu) assembler. Compare the assembly results with the results obtained using your own algorithms. Explain any differences you observe.

3 (40 pts) Task 2: Lizard assembly

For this task, you will explore a realistic *de novo* assembly using the *Scincus mitranus* (sand fish) dataset available from SRA under accession number `PRJNA1221355`. The dataset consists of accurate long reads (PacBio HiFi), less-accurate long reads (ONT), HiC (which provides long-range information), and RNA sequencing data (to identify genes and perform genome annotation).

The data is quite big and completing this task may require a lot of time and memory. You should use the Ibex cluster for this task or a powerful workstation. If you want to attempt this task on a laptop, you may subsample the fastq files you process (using `seqtk sample`) to retain only 10% of the reads. However, your assembly quality will be substantially lower than using the complete data.

3.1 Task 2.1: Genome assembly

Read the assembly article at https://www.nature.com/articles/s41576-024-00718-w. There are two assembly tools (they are more entire workflows composed of multiple error-correcting and pre-processing steps in addition to the assembly) that can do almost all steps of the assembly: Hifiasm, and Verkko. Choose one of them and assemble the *Scincus mitranus* genome.

3.2 Task 2.2: Assembly evaluation

Evaluate your assembly through the following measures.

- (8 pts) provide basic metrics, using QUAST
- (8 pts) report gene completeness, using BUSCO or asmgene
- (8 pts) report the k-mer distribution and QV score, using Merqury
- (8 pts) identify mis-assemblies, using Flagger or Inspector

(8 pts) 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?

After this step, you should have an assembly with a QV score higher than 40. However, it is possible to further increase the assembly quality (determined by QV score here) with additional refinement steps.

3.3 (Optional, 25 pts) Task 2.3: Assembly improvement

Perform additional quality control steps on the sequencing reads.

- Perform quality assessment of the raw reads using FastQC for short reads and NanoPlot or LongQC for long reads. Filter any reads you consider to be of insufficient quality. Explain which filters you applied, and why.
- Identify and remove potential contamination: Use Kraken2 or similar tools to taxonomically classify reads, and then remove reads from potential contaminants (bacterial, viral, human, etc.). Report the percentage of contamination found and removed.
- Perform adapter trimming for Illumina reads, using Trimmomatic or fastp. For ONT reads, use Porechop. For PacBio HiFi reads, no adapter trimming is necessary.

- Error correction: Illumina and PacBio HiFi reads do not usually require correction. Consider error correction for ONT reads with Herro (https://github.com/lbcb-sci/herro) or DeChat (https://www.nature.com/articles/s42003-024-07376-y). You can also use read correction built into assembly workflows like hifiasm. Additionally, you may want to (a) compare or (b) combine different error correction methods.

- Compare the assembly metrics before and after these steps. What are the effects of the steps? Document all improvements in assembly quality resulting from QC steps.

Use Mercury to determine the QV score. We will keep a ranking of the assembly quality for all students. The top three assemblies (ranked by QV score) will receive 10 additional points (35 pts total for this task).

3.4 (Optional, 50 pts) Task 2.4: Genome annotation

After obtaining a high-quality genome assembly, perform genome annotation using the pre-processed PacBio IsoSeq data (FASTQ files) from eye and liver tissues.

This task can be completed as a group (no upper limit on group size, and all group members will receive the extra credit).

- IsoSeq data preparation and quality evaluation: compute basic quality statistics of pre-processed IsoSeq FASTQ files, including read length and read quality distribution, detecting contamination (using Kraken2 or similar), test transcript completeness (check the presence of poly-A tails); cluster the isoforms (using a tool like cd-hit) and analyze cluster sizes; align reads to the genome you have previously assembled; identify potential chimeric alignments. Provide a report that includes summary statistics (total reads, mean length, N50, etc.), read length distribution plot, transcript completeness assessment, mapping rate to the genome, potential contamination assessment, number of unique transcript clusters, and tissue-specific transcript distributions (differences between eye and liver)

- Gene prediction: Use BRAKER (https://github.com/Gaius-Augustus/BRAKER) to build a gene prediction model. Make sure you use the version that uses IsoSeq data. Apply the gene prediction model on the assembled genome. Report the number and types of genes, and generate a GFF3 file with the predicted genes.

- Functional annotation: Assign functions to all predicted genes, using three different methods: (1) assign functions by sequence similarity to known genes, using BLASTP and the UniProt database; (2) assign functions using InterProScan; (3) assign Gene Ontology (GO) functions using a prediction method like DeepGO.

4 Collaboration policy

The following collaboration policies apply to this assignment:

- **Tasks 1.1–2.3**: These must be completed individually. You may discuss concepts, algorithms, and troubleshooting approaches with classmates, but all code implementation, analysis, and report writing must be your own original work. Cite resources you use, and list students you discussed your solutions with.
- **Task 2.4 (Optional genome annotation)**: This task may be completed as a group with no upper limit on group size. All group members will receive the same extra credit. Submit one annotation report per group with all group members' names clearly listed.

Sharing of code, analysis results, or report text for the individual components is not permitted.

You can use LLMs to assist in coding or writing as well as for data interpretation. If you use an LLM, include its name and version. Clearly describe the role the LLM had in generating your report: state the kind of prompt you used (for code: help with individual functions or parts of functions, help with complex tasks, or help to solve the entire assignment/task); and state the method you used to design your prompts (if any). Also state how you used the output of the LLM: did you include it verbatim (in your code, or in your report), or did you use it as inspiration but the writing is your own, or both?

If you use an LLM for *any* part of the assignment, you *must* state how you validated the correctness of the output (and you must not use an LLM to write this part).

5 Submission format

Submit your assignment through the course platform with the following components:

1. **Code repository**: A GitHub or GitLab repository containing:
    - Source code for all implemented algorithms with documentation
    - Executable scripts used for evaluation and analysis
    - README with clear instructions for reproducing your results
2. **Written report**: A single PDF document containing:
    - Brief description of the algorithms you implemented
    - Assembly results and analysis from all tasks
    - Figures showing visualizations and comparisons
    - Evaluation metrics with proper interpretation
    - Discussion of challenges encountered and solutions applied (e.g., time or space requirements)
3. **Assembly files**: Include links to your assembled genome and annotation files. Due to file size constraints, you can provide an Ibex pathname (but make sure it is readable to other users).
4. **GFF files**: For Task 2.4 (optional genome annotation), submit a separate additional report detailing the annotation process and provide the annotation results in GFF format if completed as a group.